Concept2

PRELIMINARY

Performance Monitor Generation III (PM3)
Communications Interface Definition

Filename: Concept2 PM3 Communication Interface Definition 007.doc

Revision: 0.07
9/9/04 2:23 PM

Concept2
105 Industrial Park Drive
Morrisville, VT 05661
802-888-5226 (Voice)
802-888-6331 (Fax)
rowing@concept2.com

Table of Contents

List of Figures

List of Tables

Purpose and Scope

This document contains the communications interface definition for the Concept2 PM3. The PM3 is the performance monitor for the indoor rower providing the ability to communicate with a host PC utilizing Universal Serial Bus (USB) media.

Feedback on this document and the SDK itself should be provided via the protected forum (http://concept2.ipbhost.com) for software developers. Your protected forum password should have already been provided to you. If not, please contact Scott Hamilton (rowing@concept2.com) at Concept2.

Document History

Table 1 - Document Modification History

Edit Date	Engineer	Description of Modification
5/25/04	Mark Lyons	Created from Engineering Notes, rev 0.01
6/09/04	Mark Lyons	Updated, rev 0.02
6/10/04	Mark Lyons	Released for review
6/11/0	Mark Lyons	Minor updates , rev 0.03
6/14/04	Mark Lyons	Updated, rev 0.04
6/29/04	Mark Lyons	Minor updates, rev 0.05
9/9/04	Mark Lyons	Minor updates and added PM3 error codes, rev 0.06

Related Documents

Table 2 - Related Documents

Document Title	Document Number - Date
CSAFE Protocol Technical Specification, V1.x	http://www.fitlinxx.com/csafe/
Concept2 PM3 Firmware Revision History	(TBD)
Concept2 PM3 Software Development Kit Revision History	(TBD)

Overview

The PM3 communication protocol is intended for use over the following physical interfaces: Universal Serial Bus (USB). The PM3 protocol is based on the CSAFE protocol that is targeted at supporting communications between physical fitness equipment and a host PC. Extensions to the CSAFE protocol are employed to provide the PM3-specific functionality not supported by the generic protocol while maintaining compatibility.

The communication protocol possesses the following basic features:

- Self-starting frame structure with data transparency over numerous physical interfaces
- Simple state machine model for master/slave interactions
- "Speak-when-spoken-to" configuration with the option for well-defined unsolicited response communication
- Point-to-point and point-to-multi-point network configurations
- Error detection
- Standardized data formats for time, distance, etc.
- Extensibility by virtue of equipment vendor identification and vendor custom command definitions

The following sections provide the communication protocol definition including the physical, data link, network, and transport layer aspects for each interface.

Data Flow

The following sections summarize the data used by a host PC for controlling, configuring, and monitoring a single PM3. The data flow is separated into data originating from the PM3 or from the host PC. The various data along with the size and update rate requirement serve to establish the commands/responses that exist within the communication protocol. Certain data may be included in more than one command/response if dictated by the data flow requirements.

Table 3 - PM3 Configuration/Data to PC

Parameter	Description	Update Rate (max)	Units (Resolution)	Precision	Bytes	CSAFE Command
Slave Status	Slave status	Per Request	N/A	N/A	0	CSAFE_GETSTATUS_CMD
Workout Duration	Work time duration of workout	10 Hz	N/A	HMS[4]	3	CSAFE_GETTWORK_CMD
Horizontal Distance	Work distance of workout	10 Hz[1]	Meters (1 m)	Integer	2	CSAFE_GETHORIZONTAL_CMD
Pace	Time elapsed per unit distance for a given stroke	2 Hz[2] (per stroke)	Sec/Km	Integer	2	CSAFE_GETPACE_CMD
Power	Power generated based on the pace per stroke	2 Hz (per stroke)	Watts (1 w)	Integer	2	CSAFE_GETPOWER_CMD
Accumulated Calories	Accumulated calories burned	2 Hz (per stroke)	Calories (1 cal)	Integer	2	CSAFE_GETCALORIES_CMD
Cadence	Strokes per minute for per stroke	2 Hz (per stroke)	Strokes/Min (1 stroke)	Integer	1	CSAFE_GETCADENCE_CMD
Current Heart Rate	Current heart beats per minute	1 Hz	Beats/Min (1 beat)	Integer	1	CSAFE_GETHRCUR_CMD
Product Version	Manufacturer ID, CID, Model, HW Version, Application FW Version	Per Request	Numeric	Integer		CSAFE_GETVERSION_CMD
HW Serial Number	Hardware serial number	Per Request	ASCII	Integer	9	CSAFE_GETSERIAL_CMD
User ID Number	User identifier number	Per Request	Numeric	Integer	3 – 5[3]	CSAFE_GETID_CMD
Capabilities	Device capabilities Protocol: 1. Max Rx Frame bytes 2. Max Tx Frame bytes 3. Min Interframe gap (msec.) Power: Not Applicable (all zeroes returned)	Per Request	Numeric	Integer	3 – 11[5]	CSAFE_GETCAPS_CMD

	Text: Not Applicable (all zeroes returned)					
Error Code	Last error code (see User Programmer's Guide) latched and cleared after reading	Per Request	Enumeration	Integer	3	CSAFE_GETERRORCODE_CMD
PM3 Specific Commands						
Work Time	Work time duration of workout (high resolution) **Note: this is the time seen on the display, not necessarily the elapsed time**	10 Hz	Seconds (.01 sec)	Integer	4	CSAFE_SETUSERCFG1_CMD + CSAFE_PM_GET_WORKTIME
Work Distance	Work distance of workout **Note: this is the distance seen on the display, not necessarily the accumulated distance**	10 Hz	Meters (1 m)	Integer	4	CSAFE_SETUSERCFG1_CMD + CSAFE_PM_GET_WORKDISTANCE
Drag Factor	Drag factor	2 Hz (per stroke)	N-M-Sec^2	Integer	1	CSAFE_SETUSERCFG1_CMD + CSAFE_PM_GET_DRAGFACTOR
Stroke State	State of stroke logic: 0. Waiting for wheel to reach min. speed 1. Waiting for wheel to accelerate 2. Driving 3. Dwelling after drive 4. Recovery Note: Catch would be the transition from recovery to driving. End-of-stroke would be the transition from driving to dwelling after drive	100 Hz	Enumeration	Integer	1	CSAFE_SETUSERCFG1_CMD + CSAFE_PM_GET_STROKESTATE
Workout Type	Workout Type:	Per Workout	Enumeration	Integer	1	CSAFE_SETUSERCFG1_CMD +

	0. Just Row/no splits 1. Just Row/splits 2. Fixed Distance/no splits 3. Fixed Distance/splits 4. Fixed Time/no splits 5. Fixed Time/splits 6. Fixed Time Interval 7. Fixed Distance Interval 8. Variable Interval					CSAFE_PM_GET_WORKOUTTYPE

Notes:

1. At 2000 rpm of flywheel (2000 rpm/60 seconds/m x 3 tics/revolution x 1/12.93 tics/meter = 7.7 meters/second)
2. At 100 strokes/min
3. Dependent on configuration
4. Hours/Minutes/Seconds in byte format
5. Dependent on capability code

Table 4 - PC Configuration/Data to PM3

Parameter/ Function	Description	Update Rate (max)	Units (Resolution)	Precision	Bytes	CSAFE Command
CSAFE Machine State	Sets PM3 to one of the CSAFE machine states: 0. Error 1. Ready 2. Idle 3. Have ID 4. <unassigned> 5. In Use 6. Paused 7. Finished 8. Manual 9. Offline	Per Command	N/A	N/A	0	CSAFE_GOREADY_CMD, CSAFE_GOIDLE_CMD, CSAFE_GOHAVEID_CMD, CSAFE_GOINUSE_CMD, CSAFE_GOFINISHED_CMD
CSAFE Machine Reset	Reset CSAFE state machine and related parameters	Per Command	N/A	N/A	0	CSAFE_RESET_CMD
User ID Digits	Number of user ID digits to accept (2 – 5)	Per Command	N/A	Integer	1	CSAFE_IDDIGITS_CMD
Bad User ID	Invalid User ID	Per Command	N/A	N/A	0	CSAFE_BADID_CMD
Time of Day	Time of day	Per Command	N/A	HMS[1]	3	CSAFE_SETTIME_CMD
Date	Date	Per Command	N/A	YMD[2]	3	CSAFE_SETDATE_CMD
State Timeout	Timeout period for exiting certain states	Per Command	Seconds (1 sec)	Integer	1	CSAFE_SETTIMEOUT_CMD
Workout Time	Workout time goal	Per Workout	N/A	HMS	2	CSAFE_SETTWORK_CMD
Horizontal Distance	Horizontal distance goal	Per Workout	Units Specifier	Integer	2	CSAFE_SETHORIZONTAL_CMD
Power Target	Power goal	Per Workout	Watts (1 w)	Integer	2	CSAFE_SETPOWER_CMD
Program	Programmed/ pre-stored workouts[3]: 0. Programmed 1. Standard List #1 2. Standard List #2	Per Workout	Enumeration	Integer	2	CSAFE_SETPROGRAM_CMD

	3. Standard List #3 4. Standard List #4 5. Standard List #5 6. Custom List #1 7. Custom List #2 8. Custom List #3 9. Custom List #4 10. Custom List #5 11. Favorites List #1 12. Favorites List #2 13. Favorites List #3 14. Favorites List #4 15. Favorites List #5					
User Information	User information including: 1. Weight (0 – 999 lbs/ 0 – 454 kg) 2. Age (0 – 255) 3. Gender (0: None, 1: Male, 2: Female)	Per Command	Weight (0.25 lb/0.125 kg) Age (1 year) Gender (Enum)	Integer	5	CSAFE_SETUSERINFO_CMD
PM3 Specific Commands						
Split Duration Time	Time duration of a split	Per Workout	Seconds (.01 sec)	Integer	4	CSAFE_SETUSERCFG1_CMD + CSAFE_PM_SET_SPLITDURATION
Split Duration Distance	Distance duration of a split	Per Workout	Meters (1 m)	Integer	4	CSAFE_SETUSERCFG1_CMD + CSAFE_PM_SET_SPLITDURATION

Notes:

1. Hours/Minutes/Seconds in byte format
2. Year/Month/Day in byte format
3. "Programmed" workout takes either workout time or horizontal distance goals to set-up a "fixed time" or "fixed distance" workout; "Favorites #1 – 5" are only available if a logcard is present. "Standard List" workouts defined in

Protocol Framework

In the CSAFE protocol, communication between the master and the slave(s) device is accomplished using two basic frame types: standard frame and extended frame. The standard frame provides no provisions for slave-to-slave communication or multi-drop network configurations, as device addressing is implicit. The PM3 application requires explicit device addressing for numerous scenarios (as provided by the extended frame format) so that both frame types will be handled for our implementation. In general, the slave device only speaks when responding to a master's request. Certain exceptions may be made in very specific circumstances.

The extended frame is defined as stream of bytes with the structure shown in Figure 2. Note that the standard and extended frames are identical with the exception of the frame-unique start flag and the device address information. The start flags and stop flag are unique values used to delineate the frame and, therefore, cannot appear in the frame contents or the checksum. A byte-stuffing technique is employed to ensure that these unique bytes do not occur elsewhere in the frame. A checksum is included in the frame to allow both the master and slave devices to verify the integrity of the "Frame Contents". Neither an acknowledgement (ACK) nor negative acknowledgement (NAK) at the frame level is an integral part of the protocol.

Figure 1 - Standard Frame Format

Standard Start Flag	Frame Contents	Checksum	Stop Flag

Figure 2 - Extended Frame Format

Extended Start Flag	Destination Address	Source Address	Frame Contents	Checksum	Stop Flag

Frame Structure

The frame structure is a stream of bytes with a unique start byte, optional addressing, frame contents (e.g., commands and responses), a checksum and a unique stop byte. The unique start and stop byte values are shown in Table 6. In order to ensure that these start and stop values do not appear anywhere in the frame, the master and slave devices perform "byte-stuffing" and "byte-unstuffing" on the byte stream (i.e., frame contents including extended frame addresses and checksum). This technique can be performed "on the fly" without impacting the data stream buffering requirements since the extra bytes only exist on the data link.

The extended frame addressing rules are summarized in Table 5.

Table 5 - Extended Frame Addressing

Address	Description
0x00	PC Host (master)
0x01 – 0xFC	<unassigned>
0xFD	Default slave address
0xFE	Reserved for expansion
0xFF	"Broadcast" accepted by all slaves

The "byte-stuffing" algorithm simply substitutes two bytes for each of the unique bytes listed in Table 6. The unique Byte Stuffing Flag is followed by a 0x00, 0x01, 0x02, or 0x03 as shown in Table 7 depending on the byte being replaced. The impact of this technique on the data link is that the frame size could increase in size by a factor of two in the worst case.

Table 6 - Unique Frame Flags

Description	Value
Extended Frame Start Flag	0xF0
Standard Frame Start Flag	0xF1
Stop Frame Flag	0xF2
Byte Stuffing Flag	0xF3

Table 7 - Byte Stuffing Values

Frame Byte Value	Byte-Stuffed Value
0xF0	0xF3, 0x00
0xF1	0xF3, 0x01
0xF2	0xF3, 0x02
0xF3	0xF3, 0x03

The frame beginning and end are designated by the unique Start and Stop bytes. If a Start or Stop byte is missed, the frame is discarded and frame resynchronization occurs at the beginning of the next frame. Once a full frame is received and all "byte-unstuffing" is performed, a one-byte checksum is computed with byte-by-byte XORing of the frame contents to verify frame integrity. The frame definition does not explicitly place any limits on the frame length. Because the entire frame contents must be buffered before computing the checksum, memory resources on the slave devices typically establish the restrictions on frame length. For CSAFE protocol compatibility, the following frame length restrictions are invoked for the PM3 USB physical link:

- A maximum frame size of 96 bytes including start/stop flags, checksum and byte stuffing
- All flow control handled natively as part of USB

Frame Contents

The frame protocol transports frame content data consisting of both commands and responses. The only restrictions on the frame contents relate to length of frame and the requirement that individual commands/responses do not straddle a frame boundary (i.e., no partial commands/responses within a frame). The following sections detail the command and response formats.

Command Format

All commands have one of two basic formats: long command or short command. Long commands are those including command data while short commands are command only. The command is represented by a single byte with the command address space partitioned equally (i.e., long commands have MS bit clear and short commands have MS bit set). Figure 3 and Figure 4 illustrate the long and short command formats, respectively.

Figure 3 - Long Command Format

Long Command	Data Byte Count	Data

Figure 4 - Short Command Format

Short Command

In the long command format, the Long Command and Data Byte Count fields are single bytes. The Data Byte Count field determines the Data field size. The short command format consists solely of the single byte Short Command. Table 8 summarizes the command field types for both the long and short commands. Note that the command formats allows a long command with a Data Byte Count of 0 and no bytes in the Data field. The virtue of the Data Byte Count field in the long command is to allow slave devices to handle unrecognized commands by merely disregarding the command and its data while continuing to process succeeding commands within the same frame.

Table 8 - Command Field Types

Description	Size (Bytes)	Value
Long Command	1	0x00 – 0x7F
Short Command	1	0x80 – 0xFF
Data Byte Count	1	0 - 255
Data	Variable	0 - 255

Multiple complete commands can be included in a single frame, but no partial commands or responses are allowed. Sending a frame consisting of multiple commands to a slave device results in a frame consisting of multiple command responses.

Response Format

All responses have the same Frame Contents format as shown in Figure 5.

Figure 5 - Response Frame Contents Format

Status	Command Response Data

Figure 6 - Individual Command Response Format

Identifier	Data Byte Count	Data

Table 9 - Response Field Types

Description	Size (Bytes)	Value
Status	1	0x00 – 0x7F
Command Response Data	Variable	0 - 255
Identifier	1	0x00 – 0xFF
Data Byte Count	1	1 - 255
Data	Variable	0 - 255

PM3 Manufacturer Information

Table 10 summarizes the Concept2 PM3 product-specific information.

Table 10 - CSAFE Concept2 PM3 Information

Product Information	Description
Manufacturer ID	22

Class Identifier	2
Model	3
Maximum Frame Length	96 Bytes
Minimum Inter-frame Gap	50 msec.

PM3 Extensions

The PM3 extensions to the frame protocol involve utilizing one pre-defined custom command that serves as a "wrapper" for additional PM3-specific commands. The one command is defined in Table 11. The one custom command wrapper is used to expand the CSAFE command set for additional configuration and data operations. See Appendix A for a detailed explanation of the command wrapper implementation.

Table 11 - PM3-Specific CSAFE Command Wrappers

Command Name	Command Identifier
CSAFE_SETUSERCFG1_CMD	0x1A

Additional PM3 proprietary extensions to the frame protocol involve utilizing four commands added to the existing CSAFE protocol command set that serve as "wrappers" for the PM3 command set. The four commands are defined in Table 12. The four command wrappers are used to partition the PM3 command set space into "push" (i.e., set) and "pull" (i.e., get) operations for configuration and data. The use of these command wrappers allow the PM3 to support existing CSAFE protocol commands while introducing PM3 proprietary commands only accessible via the command set extension. These commands are not accessible via the "public" interface and require special "authentication" with the PM3 to function.

Table 12 - PM3 Proprietary CSAFE Command Wrappers

Command Name	Command Identifier
CSAFE_SETPMCFG_CMD	0x76
CSAFE_SETPMDATA_CMD	0x77
CSAFE_GETPMCFG_CMD	0x7E
CSAFE_GETPMDATA_CMD	0x7F

Protocol Layer Definition

Universal Serial Bus

Physical Layer

USB Version 1.10 operating at full speed (12 Mb/s).

Table 13 - USB Series B Receptacle Mechanical

Table 14 - USB Series B Receptacle Connector Pin-out

Pin #	Signal Name
1	VBUS (+5V)
2	DATA-
3	DATA+
4	GND

Data Link Layer

In general, USB transactions consist of

- Token Packet (Header defining what it expects to follow), an
- Optional Data Packet, (Containing the payload) and a
- Status Packet (Used to acknowledge transactions and to provide a means of error correction)

USB is a host centric bus. The host initiates all transactions. The first packet, also called a token is generated by the host to describe what is to follow and whether the data transaction will be a read or write and what the device's address and designated endpoint is. The next packet is generally a data packet carrying the payload and is followed by a handshaking packet, reporting if the data or token was received successfully, or if the endpoint is stalled or not available to accept data.
Data on the bus is transmitted LS bit first. USB packets consist of the following fields,

Sync
All packets must start with a sync field. The sync field is 8 bits long at low and full speed or 32 bits long for high speed and is used to synchronize the clock of the receiver with that of the transmitter. The last two bits indicate where the PID fields starts.

PID
PID stands for Packet ID. This field is used to identify the type of packet that is being sent. The following table shows the possible values.

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

There are 4 bits to the PID, however to insure it is received correctly, the 4 bits are complemented and repeated, making an 8-bit PID in total. The resulting format is shown below.

PID_0 PID_1 PID_2 PID_3 $nPID_0$ $nPID_1$ $nPID_2$ $nPID_3$

- **ADDR**

  The address field specifies which device the packet is designated for. Being 7 bits in length allows for 127 devices to be supported. Address 0 is not valid, as any device which is not yet assigned an address must respond to packets sent to address zero.

- **ENDP**

  The endpoint field is made up of 4 bits, allowing 16 possible endpoints. Low speed devices, however can only have 2 additional endpoints on top of the default pipe. (4 endpoints max)

- **CRC**

  Cyclic Redundancy Checks are performed on the data within the packet payload. All token packets have a 5 bit CRC while data packets have a 16 bit CRC.

- **EOP**

  End of packet. Signaled by a Single Ended Zero (SE0) for approximately 2 bit times followed by a J for 1 bit time.

USB has four different packet types. Token packets indicate the type of transaction to follow, data packets contain the payload, and handshake packets are used for acknowledging data or reporting errors and start of frame packets indicate the start of a new frame.

- **Token Packets**

  There are three types of token packets,

  - **In** - Informs the USB device that the host wishes to read information.
  - **Out** - Informs the USB device that the host wishes to send information.
  - **Setup** - Used to begin control transfers.

  Token Packets must conform to the following format,

  Sync PID ADDR ENDP CRC5 EOP

- **Data Packets**

  There are two types of data packets each capable of transmitting up to 1024 bytes of data.

  - **Data0**
  - **Data1**
  -

  High Speed mode defines another two data PIDs, DATA2 and MDATA.

  Data packets have the following format,

  Sync PID Data CRC16 EOP

  - Maximum data payload size for low-speed devices is 8 bytes.
  - Maximum data payload size for full-speed devices is 1023 bytes.

- Maximum data payload size for high-speed devices is 1024 bytes.
- Data must be sent in multiples of bytes.

- **Handshake Packets**

  There are three types of handshake packets which consist simply of the PID

  - **ACK** - Acknowledgment that the packet has been successfully received.
  - **NAK** - Reports that the device temporary cannot send or received data. Also used during interrupt transactions to inform the host there is no data to send.
  - **STALL** - The device finds its in a state that it requires intervention from the host.
  -

  Handshake Packets have the following format,

  Sync PID EOP

- **Start of Frame Packets**

  The SOF packet consisting of an 11-bit frame number is sent by the host every 1ms ± 500ns on a full speed bus.

  Sync PID Frame Number CRC5 EOP

Specifically, the PM3 enumerates itself as a Human Interface Device (HID) with a control endpoint and two interrupt endpoints (IN/OUT).

Table 15 - PM3 USB Definitions

Parameter	Description
Bus Specification	USB 1.10
Bus Speed	Full-speed (12 Mbits/sec)
Control Endpoint Max Pkt Size	8 bytes
Device Description	Bus powered (98 mA max), 1 interface configuration (0)
Interface Description	Human Interface Device (HID)
Manufacturer string	“Concept2”
Product string	“Concept2 Performance Monitor 3 (PM3)”
Endpoints	IN: Interrupt/EP3/polling rate: 4 msec. OUT: Interrupt/EP4/polling rate: 1 msec.
Reports	ID #1 – 20 bytes + 1 byte report ID ID #2 – 120 bytes + 1 byte report ID

Network and Transport Layers

The CSAFE protocol provides both the network and transport layer functionality over USB. The packetization, integrity checking, and node addressing is supported by the protocol. The frame size has been increased to 96 byte from the 32 bytes specified for the asynchronous serial interface.

Protocol Features

Default Configuration

Individual manufacturers specify certain protocol parameters (e.g., timeouts, auto response behavior, etc.). Table 16 summarizes the protocol defaults for the PM3. Note that certain parameters listed in Table 16 cannot be changed (refer to the section on Unsupported Features for additional information).

Table 16 - PM3 CSAFE Protocol Defaults

Parameter	Default Value	Comments
HaveID State Transition Timeout	10 seconds	This timeout (settable via the cmdSetTimeout command) defines the delay between entering the HaveID state and transitioning back to the Idle state
Inactivity During InUse State Timeout	6 seconds	This timeout defines the duration of inactivity during the InUse state (once the workout has begun) before entering the Paused state
Inactivity During Pause State Timeout	220 seconds	This timeout defines the duration of inactivity during the Paused state (once the workout has begun) before entering the Finished state
Units Type	Metric	Metric units only
User ID Digits	5	Five-digit user ID (settable via the cmdIDDigits from 2 – 5 digits)
User ID	0 0 0 0 0	
AutoUpload Byte	0x10	flgAutoStatus: Disabled (**cannot be changed**) flgUpStatus: Disabled (**cannot be changed**) flgUpList: Disabled (**cannot be changed**) flgAck : Enabled (**cannot be changed**) flgExternControl: Disabled (**cannot be changed**)
Serial Number Digits	9	Number of digits in serial number response
PM3-specific Commands	All states	These commands are accessible in all slave states

Unsupported Features

Individual manufacturers also determine which protocol features will not be supported by their equipment. Table 17 summarizes the unsupported protocol features and the deviations from other features. In addition, the status of implementation for each CSAFE command is included in Appendix A.

Table 17 - PM3 Unsupported CSAFE Protocol Features

Feature	Comments
AutoStatus Enable	No unsolicited status uploads
UpList Enable	No unsolicited command list uploads
Ack Disable	All commands will be responded to by a least a status byte
Text Messaging	No text messaging functions
Set User Information	Not setting user weight, age and gender
Get User Information	User weight is fixed at 175 lbs, age and gender not supported
Finished State Timeout	No Finished state timeout is employed to cause a transition back to the Idle state; when a user hits the MENU/BACK to conclude viewing a finished workout result or terminate a workout in progress, the Ready state is entered
InUse State Entry	In addition to allowing entry into the InUse state from the Idle and

	HaveID states, entry from the Ready state is also allowed
Set Calories Goal	Since the PM3 allows the user to select display units (either time/meters, watts or calories), setting the workout goal using power is sufficient to define a target pace for the pace boat display for all display units.

Command and Response Definitions

Workouts

Configuring a Programmed Workout

The following steps are required to configure the workout parameters and put the PM3 screen in the proper state prior to rowing commencement:

- Set all pertinent workout parameters including either workout time (fixed time workout) or horizontal distance goal (fixed distance workout), and time/distance split duration (if not using default values). Time split duration used for "fixed time" workout and distance split duration used for "fixed distance" workout.
- If a power/calories goal is desired to control the paceboat, the proper goal value must be configured.
- Configure the programmed workout using the previously set workout parameters, and direct the PM3 to initialize the workout display parameters and go to the proper rowing screen in preparation for beginning the workout.

The following is a sample CSAFE command sequence for configuring a 2000m fixed piece with a split duration of 500 m and a power goal of 300 watts:

0x21 0x03 0x02 00 0x21 (CSAFE_SETHORIZONTAL_CMD, 2 x Km units specifier)
0x1A 0x07 0x05 0x05 0x80 0xF4 0x01 0x00 0x00
(CSAFE_SETUSERCFG1_CMD,
CSAFE_PM_SET_SPLITDURATION, distance, 500m)
0x34 0x03 0x2C 0x010x58 (CSAFE_SETPOWER_CMD, 300 x Watts unit specifier)
0x24 0x02 0x00 0x00 (CSAFE_SETPROGRAM_CMD, programmed workout)

Note that this complete sequence of commands can combined as follows with the same result:

0x21 0x03 0x02 00 0x21 0x1A 0x07 0x05 0x05 0x80 0xF4 0x01 0x00 0x00 0x34 0x03 0x2C 0x010x58
0x24 0x02 0x00 0x00

The following is a sample CSAFE command sequence for selecting the first standard list (predefined) workout in the PM3:

0x24 0x02 0x01 0x00 (CSAFE_SETPROGRAM_CMD, standard list workout #1)

PC-Host Dynamic Link Library (DLL) and Application Programming Interface (API)

Overview

PC-based host applications can communicate with PM3s over multiple physical interfaces using the DLLs. These include: Universal Serial Bus (USB).

The following sections describe the architecture and interface details of the DLLs.

Architecture

For User applications, the PM3 utilizes a protocol architecture based on CSAFE, a protocol that is targeted at supporting communications between physical fitness equipment and a host PC. Extensions to the CSAFE protocol are employed to provide PM3-specific functionality not supported by the generic protocol while maintaining compatibility.

The system architecture also supports other protocols in addition to CSAFE, including PM3 proprietary protocols for maintenance and system testing.

PM3 Device Discovery & Interface

(PM3DDI.DLL)
The PM3 Device Discovery & Interface DLL is central to all types of applications that communicate with PM3s. It provides a means for applications to identify, initialize and communicate with PM3s, as well as to discover the network topology upon which the PM3s reside. The Device Discovery interface centralizes the various addressing schemes used by the Media Interfaces, and exposes a unified, global address map to the application.

Incorporated within this DLL is a generic command/response engine. It relies on the other DLLs at the API layer for protocol-specific information (e.g. CSAFE protocol vs. PM3 proprietary protocol for implementing Flash download).

Media Interfaces

These interfaces are generally not called from the application layer, as they are meant to communicate between the Device Discovery Interface and the hardware specific windows drivers. They provide implementations of communications interfaces specific to the hardware and WDM layers. Each media interface has unique methods for identifying and addressing PM3s (See "***PM3 Comm Protocol xxx.doc***" for more information on address discovery and assignment). It is up to the Device Discovery interface to consolidate these unique addressing schemes into a unified address mapping.

USB Interface

(RPPM3USB.DLL)
This media interface implements a Human Interface Device (HID) class connection with the PM3s over a USB connection. The USB HID Class enumeration process performed by the built-in WDM functions provides unique addressing information for each connected PM3.

802.xx Interface (wireless or wired network)

(tbd)

Asynchronous Interface (RS232/485)

(tbd)

CSAFE+ Command Interface

(RPPM3CSAFE.DLL, RPPM3CSAFE.INI, RPPM3CSAFE.LIB)
The CSAFE+ Command Interface DLL exposes CSAFE commands to the application and provides the proprietary protocol-specific commands/responses that allow the implementation of the CSAFE protocol over the PM3 network. Extensions to the CSAFE protocol (hence the name CSAFE+) provide additional

proprietary functionality specific to PM3s. These proprietary commands are only available if the host PC has performed the prescribed “authentication” process with the PM3.

At run-time, the DLL reads in an initialization text file that contains the definition and format of all of the supported PM3 proprietary CSAFE commands/responses. When a command is issued by the calling application, the DLL indexes the appropriate command/response object, and passes it to the PM3 Interface. The PM3 Interface sends the command via the correct media interface, receives the response data, and then returns it back to the CSAFE DLL, which returns the data to the calling application.

This initialization file architecture allows new commands to be implemented without having to recompile the DLL code. New commands can be implemented using a text editor. Alternatively, giving this power and flexibility to the user has its’ dangers, as incorrect modification of the .INI file can lead to unpredictable behavior of the DLL.

User Application API

The PM3 User Application Programming Interface (API) is based upon and fully supports the CSAFE protocol (see http://www.fitlinxx.com/csafe/specification.htm). The PM3 User API is comprised of two DLLs: one is utilized to provide discovery functions (Version #'s, PM3 device discovery, addressing and status), and the other is used to implement the CSAFE command set, including an extended version of CSAFE to handle PM3 proprietary functionality.

The DLL functions are detailed in the sections below.

PM3 Device Discovery & Interface API

(PM3DDI.DLL, PM3DDI.H, PM3DDI.LIB)

Data Type Definitions

The following data types are utilized in this DLL:

unsigned char	UINT8
unsigned short	UINT16
unsigned long	UINT32
char	INT8
short	INT16
long	INT32
unsigned char	BOOLEAN
float32	FLOAT32
float64	FLOAT64
INT16_T	ECODE_T

Function Reference

tkcmdsetDDI_init

About: Initializes the DLL error code interface and media interfaces (e.g. USB, 802.11, Async).

Inputs: None

Outputs: None

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise

Function Prototype:

```
PM3CMD_API ERRCODE_T tkcmdsetDDI_init(void);
```

tkcmdsetDDI_shutdown

About: Shuts down the Command Set Toolkit functions.

Inputs: UINT16_T port Communication port to use

Outputs: None

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise

Function Prototype:

```
PM3CMD_API ERRCODE_T tkcmdsetDDI_shutdown(UINT16_T port);
```

tkcmdsetDDI_discover_pm3s

About: Discover all PM3 devices connected to the PC via various media interfaces. Create a PM3 device map that correlates consecutive unit identifiers to each device port number and media interface location. Note that the calling function provides the starting address of the unit ID.

Inputs:	INT8_T	*product_name	Name of product to discover
	UINT16_T	starting_address	Address of first unit
Outputs:	UINT16_T	*num_units	Number of devices found
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Function Prototype:

PM3CMD_API ERRCODE_T tkcmdsetDDI_discover_pm3s(INT8_T *product_name , UINT16_T starting_address, UINT16_T *num_units);

tkcmdsetDDI_fw_version

About: Reads the firmware version information from the PM3

Inputs:	UINT16_T	unit_address	Address of PM
Outputs:	UINT8_T *	ver_ptr	FW version string stored at this location
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Function Prototype:

PM3CMD_API ERRCODE_T tkcmdsetDDI_fw_version(UINT16_T unit_address, UINT8_T * ver_ptr);

tkcmdsetDDI_hw_version

About: Reads the hardware version information from the PM3

Inputs:	UINT16_T	unit_address	Address of PM
Outputs:	UINT8_T *	ver_ptr	HW version string stored at this location
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Function Prototype:

PM3CMD_API ERRCODE_T tkcmdsetDDI_hw_version(UINT16_T unit_address, UINT8_T * ver_ptr);

tkcmdsetDDI_serial_number

About: Reads the serial number information from the PM3

Inputs:	UINT16_T	unit_address	Address of PM

Outputs: UINT8_T * ser_ptr — Serial number string stored at this location

Returns: ERRCODE_T ecode — Zero if successful; Error code otherwise

Function Prototype:

```
PM3CMD_API ERRCODE_T tkcmdsetDDI_serial_number(UINT16_T  unit_address, UINT8_T
* ser_ptr);
```

tkcmdsetDDI_special

About: Performs special operations based on the command

Inputs: UINT16_T unit_address — Address of PM
UINT16_T cmd — Special command to execute
UINT32_T in_data — Value to send with command

Outputs: UINT32_T * out_data — Location to store value returned with response

Returns: ERRCODE_T ecode — Zero if successful; Error code otherwise

Function Prototype:

```
PM3CMD_API ERRCODE_T tkcmdsetDDI_special(UINT16_T  unit_address, UINT16_T cmd,
UINT32_T in_data, UINT32_T *out_data);
```

tkcmdsetDDI_get_dll_version

About: Returns the current version number of this DLL.

Inputs: None

Outputs: None

Returns: UINT16_T ver_info — High byte is major version number; Low byte is minor version number

Function Prototype:

```
PM3CMD_API UINT16_T tkcmdsetDDI_get_dll_version(void);
```

tkcmdsetDDI_get_error_name

About: Returns the name of the error associated with the code

Inputs: ERRCODE_T ecode — Code to be looked up
UINT16_T namelen — Maximum length of name string

Outputs: char * nameptr — Location to place name string

Returns: ERRCODE_T ecode — Zero if successful; Error code otherwise

Function Prototype:

```
PM3CMD_API ERRCODE_T tkcmdsetDDI_get_error_name(ERRCODE_T ecode,char *
nameptr,UINT16_T namelen);
```

tkcmdsetDDI_get_error_text

About: Returns the text description of the error associated with the code

Inputs:	ERRCODE_T	ecode	Code to be looked up
	UINT16_T	namelen	Maximum length of text description string
Outputs:	char *	nameptr	Location to place text description string
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Function Prototype:

PM3CMD_API ERRCODE_T tkcmdsetDDI_get_error_text(ERRCODE_T ecode,char * textptr,UINT16_T textlen);

tkcmdsetDDI_init_protocol

About: Initialize a protocol engine that will be used for device communications. This is typically called by an external DLL to setup a specific communications protocol.
NOTE: This is not typically called by the application directly.

Inputs:	ERRCODE_T	*frame_builder()	Ptr to frame builder function
	ERRCODE_T	*frame_checker()	Ptr to frame checker function
	UINT16_T	timeout	Command/response time out in MS
	UINT16_T	buffer_size	Max frame size
	UINT8_T	retries	Number of command retries

Outputs: None

Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Function Prototype:

PM3DDI_API ERRCODE_T tkcmdsetDDI_init_protocol(ERRCODE_T (*) (UINT8_T *, UINT8_T *, UINT16_T *), ERRCODE_T (*) (UINT8_T *, UINT8_T *, UINT16_T *),UINT16_T timeout, UINT16_T buffer_size, UINT8_T retries);

tkcmdsetDDI_do_protocol

About: Utilizing the protocol engine that was previously setup in tkcmdsetDDI_init_protocol(), build and send a command frame, then receive and check a response frame.
If the frame is valid, return the data.
This is typically called by an external DLL to implement a specific communications protocol.
NOTE: This is not typically called by the application directly

Inputs:	UINT16_T	port	Identifier for device
	UINT16_T	*num_cmd_bytes	Number of data bytes to send
	UINT8_T	*cmd_data	Data bytes to send
Outputs:	UINT16_T	*num_rsp_bytes	Number of received bytes
	UINT8_T	*response	Response byte

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise
Function Prototype:
PM3DDI_API ERRCODE_T tkcmdsetDDI_do_protocol(UINT16_T port,
UINT16_T *num_cmd_bytes, UINT8_T *cmd_data,
UINT16_T *num_rsp_bytes, UINT8_T *response)

CSAFE Interface API

(PM3CSAFE.DLL, PM3CSAFE.H, PM3CSAFE.LIB, PM3CSAFE.INI)

After the DLL has been initialized, the CSAFE+ command set is implemented using a single generic API function. To accomplish this, the calling application maintains individual command and response buffers (comprised of arrays of 32-bit integers), passing the pointers to these buffers along with PM device addressing and command information (see **tkcmdsetCSAFE_command** below).

Using CSAFE extended frame addressing, the DLL handles packing CSAFE frames, validating response frames, and unpacking the data into the response array.

Important: The PM3CSAFE DLL requires the PM3CSAFE.INI initialization file to be present in order to recognize PM3-proprietary CSAFE commands. The DLL reads in the data from the INI file and uses the data to determine the data types expected for commands and responses. The DLL presents data to the calling application in the return buffer according to type. For example, if a PM3 command returns six data bytes comprised as a 2-byte integer and a 4 byte float, the data will be packed in two locations in the return array (one location for each data type). The DLL also expects command data to be presented in this way, which makes it a simpler interface for the application.
The DLL takes care of formatting the CSAFE frame appropriately, so the application need only send commands and data (see examples below).

The DLL will pass standard CSAFE commands through without modification. Note however, that the calling application must pass the data unpacked, as no data type checking/packing is done by the DLL. Data count bytes must be included in standard CSAFE commands, whereas the count byte is not needed in PM3 proprietary CSAFE commands.

Important: The CSAFE DLL does not currently allow extended CSAFE frames except with PM3 proprietary commands. Extended CSAFE frames (non-PM3 proprietary) can be sent using the DDI DLL function, “tkcmdsetUSB_do_DDIcommand”.

The application program can optionally configure “streams” on selected PMs, which creates independent data pipes for unsolicited data received from the PMs. (NOT IMPLEMENTED).

COMMAND EXAMPLES:

Standard CSAFE command – get status
Command data -> 00000080 (80 = get status)
Response data -> 00000080H 00000001H 00000001H (mirrored cmd, num response bytes, data)

PM3 Proprietary CSAFE command – get ERG
Command data -> 0000007E 00000050 00000032 (7e= GETPMCFG, 50= get ERG, 32= serial#)
Response data -> 0000007EH 00000050H 000000FDH (mirrored GETPMCFG cmd, mirrored get ERG command, ERG= 253)

PM3 Proprietary CSAFE command – get work time, get projected work time, get ERG
Command data -> 0000007E 000000A0 000000A1 00000050 00000032
Response data -> 0000007EH 000000A0H 00000000H 000000A1H 00000000H 00000050H 000000FDH (GETPMCFG, CMD, DATA, CMD, DATA, CMD, DATA)

tkcmdsetCSAFE_init_protocol

About: Initializes the DLL error code interface and configures the CSAFE protocol. Uses extended frame addressing.

Inputs:	UINT16_T	timeout	Command/response timeout in MS (defaults to 1000ms (1 sec) if NULL)
Outputs:	None		
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Function Prototype:
PM3CSAFE_API ERRCODE_T tkcmdsetCSAFE_init_protocol(UINT16_T timeout);

tkcmdsetCSAFE_read_logblk8

About: Reads an 8-bit block of data from a logcard memory location.

Inputs:	UINT16_T	unit_address	Address of PM device
	UINT32_T	logcard_address	Address of logcard memory to read
	UINT32_T	byte_len	Number of bytes to be read (0 returns all)
Outputs:	UINT8_T *	val_ptr8	Location to store block of data read
	UINT32_T *	num_read	Actual number of bytes read
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Function Prototype:
PM3CSAFE_API ERRCODE_T tkcmdsetCSAFE_read_logblk8(UINT16_T unit_address, UINT32_T logcard_address, UINT32_T byte_len, UINT8_T *val_ptr8, UINT32_T *num_read);

tkcmdsetCSAFE_command

About: Sends a CSAFE command to a PM device and returns the response data. Note: the unit address is previously determined using the DiscoverPM3s function in the PM3DDI DLL.

Inputs:	UINT16_T	unit_address	Address of PM device
	UINT16_T	cmd_data_size	Size of cmd data
	UINT32_T	cmd_data[]	Command data
Outputs:	UINT16_T	*rsp_data_size	Size of rsp data
	UINT32_T	rsp_data[]	Response data

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise

Function Prototype:
PM3CSAFE_API ERRCODE_T tkcmdsetCSAFE_command(UINT16_T unit_address,
UINT16_T cmd_data_size, UINT32_T cmd_data[],
UINT16_T *rsp_data_size, UINT32_T rsp_data[]);

tkcmdsetCSAFE_get_dll_version

About: Returns the current version number of this DLL.

Inputs: None

Outputs:None

Returns: UINT16_T ver_info High byte is major version number
Low byte is minor version number

Function Prototype:
PM3CMD_API UINT16_T tkcmdsetCSAFE_get_dll_version(void);

tkcmdsetCSAFE_get_error_name

About: Returns the name of the error associated with the code

Inputs: ERRCODE_T ecode Code to be looked up
UINT16_T namelen Maximum length of name string

Outputs:char * nameptr Location to place name string

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise

Function Prototype:
PM3CMD_API ERRCODE_T tkcmdsetCSAFE_get_error_name(ERRCODE_T ecode,char * nameptr,UINT16_T namelen);

tkcmdsetCSAFE_get_error_text

About: Returns the text description of the error associated with the code

Inputs: ERRCODE_T ecode Code to be looked up
UINT16_T namelen Maximum length of text description string

Outputs:char * nameptr Location to place text description string

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise

Function Prototype:
PM3CMD_API ERRCODE_T tkcmdsetCSAFE_get_error_text(ERRCODE_T ecode,char * textptr,UINT16_T textlen);

tkcmdsetCSAFE_get_status

About: Gets the CSAFE status byte from the previous transaction.

Inputs: Nothing

Outputs: Nothing

Returns: UINT8_T slave_status CSAFE status byte

**********/

```
PM3CSAFE_API UINT8_T tkcmdsetCSAFE_get_status(void);
```

PM3 USB Interface API

(RPPM3USB.DLL, RPPM3USB.H, RPPM3USB.LIB)
The following is the API for the USB DLL. NOTE: The functions in this DLL are not typically called by the application (the API layer DDI DLL calls these functions). CALLING DIRECTLY TO THE USB DLL FUNCTIONS IS RECOMMENDED ONLY FOR ADVANCED USERS!
Note that the complete CSAFE frame must be properly formatted (including byte stuffing and checksum!) when using this direct interface to communicate with the PM3.

tkcmdsetUSB_get_dll_version()

Returns the current version number of this software.

Inputs: None

Outputs: None

Returns: UINT16_T ver_info High byte is major version number
Low byte is minor version number

tkcmdsetUSB_get_error_name

About: Returns the name of the error associated with the code

Inputs: ERRCODE_T ecode Code to be looked up
UINT16_T namelen Maximum length of name string

Outputs: char * nameptr Location to place name string

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise

tkcmdsetUSB_get_error_text

About: Returns the text description of the error associated with the code

Inputs: ERRCODE_T ecode Code to be looked up
UINT16_T namelen Maximum length of text description string

Outputs: char * nameptr Location to place text description string

Returns: ERRCODE_T ecode Zero if successful
Error code otherwise

tkcmdsetUSB_init

About: Initializes the Command Set Toolkit functions.
Opens the error .INI file, and gets port numbers of all USB HID devices that match the product name.

Inputs: INT16_T * product_name Name of USB device to open

Outputs: UINT8_T * num_found Number of devices that match name

	UINT8_T	port_list[]	Port numbers of devices that match
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

tkcmdsetUSB_set_feature

About: Sends a USB SetFeature Standard Request to the device
This is independent of the command / response protocol

Inputs:	UINT16_T	port	Communication port to use
	UINT16_T	feature	Identifier for desired feature to activate
Outputs:	None		
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

tkcmdsetUSB_do_DDIcommand

About: Send and receive a data block from the DDI interface.

Inputs:	UINT16_T	port	Identifier for device
	UINT8_T *	cmd_ptr	Location of data to send to device
	UINT16_T	timeout	Command/response timeout in ms
Outputs:	UINT8_T *	rsp_ptr	Location to store data returned by device
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

tkcmdsetUSB_status

About: Reads status information from the device

Inputs:	UINT16_T	port	Identifier for device
Outputs:	UINT32_T *	stat_ptr	Location to store status information
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

tkcmdsetUSB_fw_version

About: Reads the firmware version information from the PM3

Inputs:	UINT16_T	port	Communication port to use
Outputs:	UINT8_T *	ver_ptr	FW version string stored at this location
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

tkcmdsetUSB_hw_version

About: Reads the hardware version information from the PM3

Inputs:	UINT16_T	port	Communication port to use
Outputs:	UINT8_T *	ver_ptr	HW version string stored at this location
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

tkcmdsetDDI_serial_number

About: Reads the serial number information from the PM3

Inputs:	UINT16_T	port	Communication port to use
Outputs:	UINT8_T *	ser_ptr	Serial number string stored at this location
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

tkcmdsetUSB_shutdown

About: Shuts down the Command Set Toolkit functions

Inputs:	UINT16_T	port	Communication port to use
Outputs:	None		
Returns:	ERRCODE_T	ecode	Zero if successful Error code otherwise

Example Application Logic – CSAFE DLLs.

The following demonstrates setup and operation of the CSAFE DLLs.

```
/* Initialize the DLL and media (hardware) interfaces */
if (ecode = tkcmdsetDDI_init())
{
        /* PM3 DLL init error */
}
/* Discover and get count of all discovered PM3s.  Tell DLL to start numbering at 0. */
else if (ecode = tkcmdsetDDI_discover_pm3s(TKCMDSET_PM3_PRODUCT_NAME2, 0, &num_units))
{
        /* PM discovery error */
        tkcmdsetDDI_shutdown();
}
/* Initialize the CSAFE protocol engine.  Leave timeout at default. */
else if (ecode = tkcmdsetCSAFE_init_protocol()
{
        /* CSAFE DLL init error */
}
/* Program loop */
else
{
        /* (PM initialization goes here, using CSAFE commands) */

        while (program_is_running)
        {
                /*
                 * Example command : PM_GET_WORK_TIME
                 * Uses the CSAFE command wrapper for PM3-specfic commands.
                 *  Where:
                 *          unit_address = 0;
                 *          // (uses CSAFE_GETPMDATA_CMD)
                 *          cmd_data[] = {0x7f, 0xA0}; // (PM_GET_WORK_TIME)
                 *          cmd_size = 2;
                 */
                tkcmdsetCSAFE _command(unit_address, cmd_size, cmd_data, &rsp_size, rsp_data);
                /*
                 * Example response data:
                 *          rsp_data[] = {0x7F, 0xA0, 0xFFFFFFFF};    // (GETPMDATA id,
                                                                      // GETWORKTIME id,
                 *                                                    // work time FLOAT)
                 *          rsp_size = 3;
                 *          (notice: data is returned in the buffer according to type, i.e. the
                 *          DLL packs the 4-byte float).
                 */

                /*
                 * Example command : PM_GET_ERG, PM_GET_WORK_TIME
                 * Uses the CSAFE command wrapper for PM3-specfic commands.
                 *  Where:
                 *          unit_address = 0;
                 *          // (uses CSAFE_GETPMDATA_CMD)
                 *          cmd_data[] = {0x7f, 0x50, 0x32, 0xA0};    // (GETPMDATA id,
                                                                      // GETERG id,
```

```
                                                                    // Serial#=50
                                                                    // GETWORKTIME id)
                */
                tkcmdsetCSAFE _command(unit_address, cmd_size, cmd_data, &rsp_size, rsp_data);
                /*
                 * Example response data:
                 *      rsp_data[] = {0x7F, 0x50, 0xfd, 0xA0, 0xFFFFFFFF};          //
                                                                    // (GETPMDATA id,
                                                                    // GETERG id,
                                                                    // ERG # = 253
                                                                    // GETWORKTIME id,
                *                                                   // work time FLOAT)
                */
        }

        /* Clean up and exit */
        tkcmdset_shutdown();
}
```

Example Application Logic – USB DLL ("direct interface").

The following demonstrates calling directly into the USB DLL, bypassing the CSAFE and DDI DLLs. Note that the complete CSAFE frame must be properly formatted (including byte stuffing!) when using this direct interface to communicate with the PM3.

```
/* Initialize the USB DLL interface */
if (ecode = tkcmdsetUSB_init())
{
        /* PM3 USB DLL init error */
}
/* Initialize and discover USB PM3 devices. Update device map structure.*/
/* usb_port[] is an array of USB port numbers for all PM3's found - will get populated by this routine */
else if (ecode = tkcmdsetUSB_init(TKCMDSET_PM3_PRODUCT_NAME2, &usb_num_units, usb_port))
{
        /* USB init error – shut down DLL */
        tkcmdsetUSB_shutdown();
}
/* Program loop */
else
{
        /* (PM initialization goes here, using CSAFE commands) */

        while (program_is_running)
        {
                /*
                 * Example command : GET STATUS (uses extended frame format)
                *  Where:
                 *      port = 0;
                 *      timeout = 70; (in ms)
                 *      cmd_data[] = {0xF0, 0xFD, 0x00, 0x80, 0x80, 0xF2}; // GET STATUS
                */
                  ecode = tkcmdsetUSB_do_DDIcommand(, cmd_data, rsp_data, timeout);

                /* Example response data:
                 *      rsp_data[] = { 0xF0, 0x00, 0xFD, 0x01, 0x80, 0x01, 0x01, 0x81, 0xF2};
```

```
                */
        }

        /* Clean up and exit */
        tkcmdsetUSB_shutdown();
}
```

Appendix A

CSAFE Commands Implemented

Short Commands

Command Name	Command Identifier	Response Data
CSAFE_GETSTATUS_CMD	0x80	Byte 0: Status
CSAFE_RESET_CMD	0x81	N/A[1]
CSAFE_GOIDLE_CMD	0x82	N/A[1]
CSAFE_GOHAVEID_CMD	0x83	N/A[1]
CSAFE_GOINUSE_CMD	0x85	N/A[1]
CSAFE_GOFINISHED_CMD	0x86	N/A[1]
CSAFE_GOREADY_CMD	0x87	N/A[1]
CSAFE_BADID_CMD	0x88	N/A[1]
CSAFE_GETVERSION_CMD	0x91	Byte 0: Mfg ID Byte 1: CID Byte 2: Model Byte 3: HW Version (LS) Byte 4: HW Version (MS) Byte 5: SW Version (LS) Byte 6: SW Version (MS)
CSAFE_GETID_CMD	0x92	Byte 0: ASCII Digit 0 (MS) Byte 1: ASCII Digit 1 Byte 2: ASCII Digit 2[2] Byte 3: ASCII Digit 3[2] Byte 4: ASCII Digit 4[2] (LS)
CSAFE_GETUNITS_CMD	0x93	Byte 0: Units Type
CSAFE_GETSERIAL_CMD	0x94	Byte 0: ASCII Serial# (MS) Byte 1: ASCII Serial # Byte 2: ASCII Serial # Byte 3: ASCII Serial # Byte 4: ASCII Serial # Byte 5: ASCII Serial # Byte 6: ASCII Serial # Byte 7: ASCII Serial # Byte 8: ASCII Serial # (LS)
CSAFE_GETLIST_CMD	0x98	<Not implemented>
CSAFE_GETUTILIZATION_CMD	0x99	<Not implemented>
CSAFE_GETMOTORCURRENT_CMD	0x9A	<Not implemented>
CSAFE_GETODOMETER_CMD	0x9B	Byte 0: Distance (LSB) Byte 1: Distance Byte 2: Distance Byte 3: Distance (MSB)

		Byte 4: Units Specifier
CSAFE_GETERRORCODE_CMD	0x9C	Byte 0: Error Code (LSB) Byte 1: Error Code Byte 2: Error Code (MSB)
CSAFE_GETSERVICECODE_CMD	0x9D	<Not implemented>
CSAFE_GETUSERCFG1_CMD	0x9E	<Not implemented>
CSAFE_GETUSERCFG2_CMD	0x9F	<Not implemented>
CSAFE_GETTWORK_CMD	0xA0	Byte 0: Hours Byte 1: Minutes Byte 2: Seconds
CSAFE_GETHORIZONTAL_CMD	0xA1	Byte 0: Horizontal Distance (LSB) Byte 1: Horizontal Distance (MSB) Byte 2: Units Specifier
CSAFE_GETVERTICAL_CMD	0xA2	<Not implemented>
CSAFE_GETCALORIES_CMD	0xA3	Byte 0: Total Calories (LSB) Byte 1: Total Calories (MSB)
CSAFE_GETPROGRAM_CMD	0xA4	Byte 0: Programmed/Pre-stored Workout Number
CSAFE_GETSPEED_CMD	0xA5	<Not implemented>
CSAFE_GETPACE_CMD	0xA6	Byte 0: Stroke Pace (LSB) Byte 1: Stroke Pace (MSB) Byte 2: Units Specifier
CSAFE_GETCADENCE_CMD	0xA7	Byte 0: Stroke Rate (LSB) Byte 1: Stroke Rate (MSB) Byte 2: Units Specifier
CSAFE_GETGRADE_CMD	0xA8	<Not implemented>
CSAFE_GETGEAR_CMD	0xA9	<Not implemented>
CSAFE_GETUPLIST_CMD	0xAA	<Not implemented>
CSAFE_GETUSERINFO_CMD	0xAB	Byte 0: Weight (LSB) Byte 1: Weight (MSB) Byte 2: Units Specifier Byte 3: Age Byte 4: Gender
CSAFE_GETTORQUE_CMD	0xAC	<Not implemented>
CSAFE_GETHRCUR_CMD	0xB0	Byte 0: Beats/Min
CSAFE_GETHRTZONE_CMD	0xB2	<Not implemented>
CSAFE_GETMETS_CMD	0xB3	<Not implemented>
CSAFE_GETPOWER_ CMD	0xB4	Byte 0: Stroke Watts (LSB) Byte 1: Stroke Watts (MSB) Byte 2: Units Specifier
CSAFE_GETHRAVG_CMD	0xB5	<Not implemented>
CSAFE_GETHRMAX_CMD	0xB6	<Not implemented>
CSAFE_GETUSERDATA1_CMD	0xBE	<Not implemented>
CSAFE_GETUSERDATA2_CMD	0xBF	<Not implemented>
CSAFE_GETAUDIOCHANNEL_CMD	0xC0	<Not implemented>
CSAFE_GETAUDIOVOLUME_CMD	0xC1	<Not implemented>
CSAFE_GETAUDIOMUTE_CMD	0xC2	<Not implemented>
CSAFE_ENDTEXT_CMD	0xE0	<Not implemented>
CSAFE_DISPLAYPOPUP_CMD	0xE1	<Not implemented>
CSAFE_GETPOPUPSTATUS_CMD	0xE5	<Not implemented>

Notes:

1. No specific response data, but the status byte will be returned
2. Depends on # ID digits configuration

Example CSAFE command/response frames using standard frame:

Get status using the CSAFE_GETSTATUS_CMD command -

Command Frame: 0xF1 0x80 0x80 0xF2
Response Frame: 0xF1 0x01 0x80 0x01 0x01 0x81 0xF2

Example CSAFE command/response frames using extended frames with the host PC address of 0x00 and the default PM (Erg) address of 0xFD:

Get version information using the CSAFE_GETVERSION_CMD command -

Command Frame: 0xF0 0xFD 0x00 0x91 0x91 0xF2
Response Frame: 0xF0 0x00 0xFD 0x81 0x91 0x07 0x16 0x02 0x03 0xA4 0x01 0x84 0x03 0xA3 0xF2

Long Commands

Command Name	Command Identifier	Command Data	Response Data
CSAFE_AUTOUPLOAD_CMD[2]	0x01	Byte 0: Configuration	N/A
CSAFE_UPLIST_CMD	0x02	<Not implemented>	N/A
CSAFE_UPSTATUSSEC_CMD	0x04	<Not implemented>	N/A
CSAFE_UPLISTSEC_CMD	0x05	<Not implemented>	N/A
CSAFE_IDDIGITS_CMD	0x10	Byte 0: # of Digits	N/A
CSAFE_SETTIME_CMD	0x11	Byte 0: Hour Byte 1: Minute Byte 2: Second	N/A
CSAFE_SETDATE_CMD	0x12	Byte 0: Year Byte 1: Month Byte 2: Day	N/A
CSAFE_SETTIMEOUT_CMD	0x13	Byte 0: State Timeout	N/A
CSAFE_SETUSERCFG1_CMD[1]	0x1A	One or more PM3 specific commands	<PM3 specific command identifer(s)>
CSAFE_SETUSERCFG2_CMD	0x1B	<Not implemented>	N/A
CSAFE_SETTWORK_CMD	0x20	Byte 0: Hours Byte 1: Minutes Byte 2: Seconds	N/A
CSAFE_SETHORIZONTAL_CMD	0x21	Byte 0: Horizontal Distance (LSB) Byte 1: Horizontal Distance (MSB) Byte 2: Units Specifier	N/A
CSAFE_SETVERTICAL_CMD	0x22	<Not implemented>	N/A
CSAFE_SETCALORIES_CMD	0x23	Byte 0: Total Calories (LSB) Byte 1: Total Calories (MSB)	N/A
CSAFE_SETPROGRAM_CMD	0x24	Byte 0: Programmed or Pre-stored Workout Byte 1: <don't care>	N/A
CSAFE_SETSPEED_CMD	0x25	<Not implemented>	N/A
CSAFE_SETGRADE_CMD	0x28	<Not implemented>	N/A
CSAFE_SETGEAR_CMD	0x29	<Not implemented>	N/A
CSAFE_SETUSERINFO_CMD	0x2B	<Not implemented>	N/A

CSAFE_SETTORQUE_CMD	0x2C	<Not implemented>	N/A
CSAFE_SETLEVEL_CMD	0x2D	<Not implemented>	N/A
CSAFE_SETTARGETHR_CMD	0x30	<Not implemented>	N/A
CSAFE_SETMETS_CMD	0x33	<Not implemented>	N/A
CSAFE_SETPOWER_CMD	0x34	Byte 0: Stroke Watts (LSB) Byte 1: Stroke Watts (MSB) Byte 2: Units Specifier	N/A
CSAFE_SETHRZONE_CMD	0x35	<Not implemented>	N/A
CSAFE_SETHRMAX_CMD	0x36	<Not implemented>	N/A
CSAFE_SETCHANNELRANGE_CMD	0x40	<Not implemented>	N/A
CSAFE_SETVOLUMERANGE_CMD	0x41	<Not implemented>	N/A
CSAFE_SETAUDIOMUTE_CMD	0x42	<Not implemented>	N/A
CSAFE_SETAUDIOCHANNEL_CMD	0x43	<Not implemented>	N/A
CSAFE_SETAUDIOVOLUME_CMD	0x44	<Not implemented>	N/A
CSAFE_STARTTEXT_CMD	0x60	<Not implemented>	N/A
CSAFE_APPENDTEXT_CMD	0x61	<Not implemented>	N/A
CSAFE_GETTEXTSTATUS_CMD	0x65	<Not implemented>	N/A
CSAFE_GETCAPS_CMD	0x70	Byte 0: Capability Code	Capability Code 0x00: Byte 0: Max Rx Frame Byte 1: Max Tx Frame Byte 2: Min Interframe Capability Code 0x01: Byte 0: 0x00 Byte 1: 0x00 Capability Code 0x02: Byte 0: 0x00 Byte 1: 0x00 Byte 2: 0x00 Byte 3: 0x00 Byte 4: 0x00 Byte 5: 0x00 Byte 6: 0x00 Byte 7: 0x00 Byte 8: 0x00 Byte 9: 0x00 Byte 10: 0x00
CSAFE_SETPMCFG_CMD[3]	0x76	1 or more PM3 CSAFE commands	See PM3 proprietary commands
CSAFE_SETPMDATA_CMD[3]	0x77	1 or more PM3 CSAFE commands	See PM3 proprietary commands
CSAFE_GETPMCFG_CMD[3]	0x7E	1 or more PM3 CSAFE commands	See PM3 proprietary commands
CSAFE_GETPMDATA_CMD[3]	0x7F	1 or more PM3 CSAFE commands	See PM3 proprietary commands

Notes:

1. Used for PM3-specific functionality as command wrapper
2. Although implemented, this command currently has no affect
3. Added for PM3 proprietary functionality as command wrappers (not available for public use)

Example CSAFE command/response frame using standard frame:

Set work time goal to 7:30 with CSAFE_SETTWORK_CMD command -

Command Frame: 0xF1 0x20 0x03 0x00 0x07 0x1E 0x3A 0xF2
Response Frame: 0xF1 0x01 0x05 0x80 0x02 0x00 0x01 0xF9 0xF2

Example CSAFE PM3 proprietary command/response frame using extended frame with the host PC address of 0x00 and the default PM (Erg) address of 0xFD:

Get device protocol parameter capabilities with the CSAFE_GETCAPS_CMD command -

Command Frame: 0xF0 0xFD 0x00 0x70 0x01 0x00 0x71 0xF2
Response Frame: 0xF0 0x00 0xFD 0x81 0x70 0x03 0x60 0x60 0x032 0x41 0xF2

CSAFE PM3-Specific Commands Implemented

Short Commands

Command Name	Command Identifier	Response Data
CSAFE_PM_GET_WORKOUTTYPE	0x89	Byte 0: Workout Type
CSAFE_PM_GET_DRAGFACTOR	0xC1	Byte 0: Drag Factor
CSAFE_PM_GET_STROKESTATE	0xBF	Byte 0: Stroke State
CSAFE_PM_GET_WORKTIME	0xA0	Byte 0: Work Time (LSB) Byte 1: Work Time Byte 2: Work Time Byte 3: Work Time (MSB)
CSAFE_PM_GET_WORKDISTANCE	0xA3	Byte 0: Work Distance (MSB) Byte 1: Work Distance Byte 2: Work Distance Byte 3: Work Distance (LSB)

Example CSAFE command/response frame using standard frame:

Get higher resolution work time (2:30.85 seconds) using the CSAFE_SETUSERCFG1_CMD command wrapper with a CSAFE_PM_GET_WORKTIME command -

Command Frame: 0xF1 0x1A 0x01 0xA0 0xBB 0xF2
Response Frame: 0xF1 0x81 0x1A 0x06 0xA0 0x04 0xED 0x3A 0x00 0x00 0x6F 0xF2

Example CSAFE command/response frames using extended frames with the host PC address of 0x00 and the default PM (Erg) address of 0xFD:

Get workout type (fixed distance w/ splits) and drag factor (128) using the CSAFE_SETUSERCFG1_CMD command wrapper with a CSAFE_PM_GET_WORKOUTTYPE and CSAFE_PM_GET_DRAGFACTOR commands -

Command Frame: 0xF0 0xFD 0x00 0x1A 0x02 0x89 0xC1 0x50 0xF2
Response Frame: 0xF0 0x00 0xFD 0x01 0x1A 0x06 0x89 0x01 0x03 0xC1 0x01 0x80 0xD7 0xF2

Long Commands

Command Name	Command Identifier	Command Data	Response Data

CSAFE_PM_SET_SPLITDURATION	0x05	Byte 0: Time/Distance duration (0: Time, 128: Distance) Byte 1: Duration (LSB) Byte 2: Duration Byte 3: Duration Byte 4: Duration (MSB)	N/A

Example CSAFE command/response frame using standard frame:

Set the split duration (100 m) using the CSAFE_SETUSERCFG1_CMD command wrapper with a CSAFE_PM_SET_SPLITDURATION command -

Command Frame: 0xF1 0x1A 0x07 0x05 0x05 0x80 0x64 0x00 0x00 0x00 0xF9 0xF2
Response Frame: 0xF1 0x81 0x1A 0x01 0x0x05 0x1E 0xF2

Appendix B

PM3 Data Conversions

Watts <-> Pace

Pace is in sec/meter:

Watts = (2.8 / (pace * pace * pace))

Calories/Hr <-> Pace

Pace is in sec/meter:

Calories/Hr = (((2.8 / (pace * pace * pace)) * (4.0 * 0.8604)) + 300.0)

Pace <-> /500m Pace

Pace is in sec/meter:

Pace/500m = (pace * 500)

Appendix C

Pre-programmed workout definitions for standard list and custom list are defined below. Note that the "Custom List" and "Favorites" workouts can vary from PM3 to PM3 depending on actions taken by the user.

Standard List Workouts

Program # / Description
1 - 2000m Fixed Distance with 500m splits
2 - 5000m Fixed Distance with 1000m splits
3 - 10000m Fixed Distance with 2000m splits
4 - 30:00 Fixed Time w/ 6:00 splits

5 - 500m Fixed Distance Interval with 1:00 rest between intervals (500m/1:00r)

Custom List Workouts

Program # / Description
6 - 00:30 Fixed Time Interval w/ 00:30 rest between intervals (:30/:30r)
7 - 7 Interval Variable (1:00/1:00r, 2:00/2:00r, 3:00/3:00r, 4:00/4:00r, 3:00/3:00r, 2:00/2:00r, 1:00/1:00r)
8 - 4 Interval Variable (2000m/3:00r, 1500m/3:00r, 1000m/3:00r, 500m/3:00r)
9 - 9 Interval Variable (1:40/:20r, 1:40/:20r, 1:40/:20r, 1:40/:20r, 1:40/2:00r, 1:40/:20r, 1:40/:20r, 1:40/:20r, 1:40/:20r)
10 - 42195 Fixed Distance with 2000m splits

Appendix D

The PM3 error display format is a combination of error code and screen number as defined below:

<Error Code> - <Screen Number>

The error codes and their descriptions are provided in Table 18.

PM3 Error Codes

Table 18 - PM3 Error Code Descriptions

Internal Name	Value	Description
APMAIN_TASKCREATE_ERR	1	Operating system task creation error
APMAIN_TASKDELETE_ERR	2	Operating system task deletion error
APMAIN_VOLTSUPPLY_ERR	3	<Not Used>
APMAIN_USERKEY_STUCK_ERR	4	One or more user input keys are asserted during power-up and not released within 2 seconds
APMAIN_TASK_INVALID_ERR	5	One or more operating system tasks that should be active during normal operation are determined to be inactive
APCOMM_INIT_ERR	10	<Not Used>
APCOMM_INVALIDPW_ERR	11	Invalid interface authentication password
APLOG_INIT_ERR	20	<Not Used>
APLOG_INVALIDUSER_ERR	21	User number provided by the screen content is out of range
APLOG_USERSTATINFO_STORAGE_ERR	22	User static information not successfully stored on logcard
APLOG_USERSTATINFO_RETRIEVE_ERR	23	User static information not successfully retrieved from logcard
APLOG_USERDELETE_ERR	24	Unsuccessful deletion of user from logcard
APLOG_USERDYNAMINFO_STORAGE_ERR	25	User dynamic information not successfully stored on logcard
APLOG_USERDYNAMINFO_RETRIEVE_ERR	26	User dynamic information not successfully retrieved from logcard
APLOG_CUSTOMWORKOUT_STORAGE_ERR	27	Custom workout information not successfully stored on logcard
APLOG_CUSTOMWORKOUT_RETRIEVE_ERR	28	Custom workout information not successfully retrieved from logcard
APLOG_CUSTOMWORKOUT_INSUFFMEM_ERR	29	Insufficient logcard memory exists to store the custom workout information
APLOG_CUSTOMWORKOUT_INVALID_ERR	30	Specific custom workout information is invalid

Internal Name	Value	Description
APLOG_INVALIDCARDOPERATION_ERR	31	Screen content performed invalid logcard operation
APLOG_INVALIDUSERCARDDATA_ERR	32	<Not Used>
APLOG_INVALIDCUSTOMWORKOUT_ERR	33	Custom workout number provided by the screen content is out of range
APLOG_INVALIDWORKOUTIDENT_ERR	34	Workout type provided by the screen content is out of range
APLOG_INVALIDLISTLENGTH_ERR	35	<Not Used>
APLOG_INVALIDINPUTPARAM_ERR	36	Special function input parameter provided by screen content is invalid
APLOG_INVALIDWORKOUTNUM_ERR	37	Workout number provided by the screen content is out of range
APLOG_CARDNOTPRESENT_ERR	38	Logcard access unsuccessful because card not present
APLOG_INVALIDINTLOGADDR_ERR	39	Logcard workout log address provided by the screen content was out of range
APLOG_INVALIDLOGHDRPTR_ERR	40	Accessing the logcard workout log section was unsuccessful because some of the contents are invalid
APLOG_MAXSPLITSEXCEEDED_ERR	41	Unable to store the split/interval data in internal log memory because the maximum # of splits has been exceeded
APLOG_NODATAAVAILABLE_ERR	42	Searching for the requested logcard workout log data has returned no information
APLOG_INVALIDCARDSTRUCTREV_ERR	43	Logcard card information structure revision is invalid
APLOG_CARDOPERATIONTIMEOUT_ERR	44	Logcard operations requested by the screen content timed-out waiting for the logcard to become available
APLOG_INVALIDLOGSIZE_ERR	45	Detected invalid data set size while storing workout results to logcard
APLOG_LOGENTRYVALIDATE_ERR	46	Failure to validate workout results written to logcard
APLOG_USERDYNAMICVALIDATE_ERR	47	Failure to validate updated user dynamic data written to logcard
APLOG_CARDINFOVALIDATE_ERR	48	Failure to validate updated card information data written to logcard
APLOG_CARDACCESS_ERR	49	Unable to communicate with logcard while its status is present and valid
APPM3_INVALIDWORKOUTNUM_ERR	60	Workout number provided by screen content or host PC for configuring the PM3 is out of range
APPM3_NOPLOTDATA_ERR	61	No pace plot data available for collection by the host PC
APPM3_INVALIDMFGINFO_ERR	62	Manufacturing information structure stored in non-volatile memory does not pass its integrity check
APPM3_INVALIDCALINFO_ERR	63	Calibration information structure stored in non-volatile memory does not pass its integrity check

Internal Name	Value	Description
APHEADER_INVALIDFONTHDR_ERR	80	Font information header structure stored in Flash memory does not pass its integrity check
APHEADER_INVALIDSCRNHDR_ERR	81	Screen content information header structure stored in Flash memory does not pass its integrity check
AP_NETWORK_GENESISMODE_ERR	1230	
AP_NETWORK_PRIMFWDWNLOAD_ERR	1250	
AP_NETWORK_READPDA_ERR	1251	
AP_NETWORK_LOADBLOCKS_ERR	1252	
AP_NETWORK_VERIFYBLOCKS_ERR	1253	
AP_NETWORK_APPLYPDA_ERR	1254	
AP_NETWORK_RUNFW_ERR	1255	
AP_NETWORK_RESET_ERR	1256	
AP_NETWORK_PRESENT_ERR	1257	
AP_NETWORK_SECFWDWNLOAD_ERR	1350	
AP_NETWORK_DRIVERINIT_ERR	1400	
AP_NETWORK_NETWORKINIT_ERR	1401	
AP_NETWORK_UDPSERVERINIT_ERR	1402	
AP_NETWORK_COMMSQUALITY_ERR	1500	
AP_NETWORK_PACKETSTATS_ERR	1501	
AP_NETWORK_POWERMANAGECFG_ERR	1502	
TKCMDPR_INVALIDMSGTYPE_ERR	120	<Not Used>
TKCMDPR_INVALID_CMD_ERR	121	Command received from host PC is invalid
TKCMDPR_INVALID_CMD_ADDR_ERR	122	CSAFE command frame address is invalid
TKCMDPR_INVALID_DEST_ADDR_ERR	123	CSAFE command frame routing destination address is invalid
TKCMDPR_INVALID_DEST_INTF_ERR	124	CSAFE command/response frame routing table entry has conflicting destination interface with existing entry
TKCMDPR_INVALID_INTF_ERR	125	CSAFE command/response processing specified an invalid communication interface
TKCMDPR_ROUTE_TABLE_FULL_ERR	126	Command/response frame routing table full
TKCMDPR_UNAUTHORIZED_CMD_ERR	127	Command received from host PC is not supported in the current PM3 operating mode
TKCMDPR_REFUSE_CMD_ERR	128	Broadcast command from host PC refused in the current PM3 operating mode
TKCMDPR_INVALID_RSP_ERR	129	Command frame received from host PC is valid but at least one of the individual commands within the frame is invalid/unsupported so

Internal Name	Value	Description
		no response is generated
TKDATALOG_INIT_ERR	130	Data logging toolkit functions not initialized
TKDATALOG_DEVICE_INVALID_ERR	131	Selected data logging device invalid
TKDATALOG_CARD_INIT_ERR	132	Failure of data logging device to pass integrity check
TKDATALOG_DEVICE_SIZE_ERR	133	Data logging devices size is not supported
TKDATALOG_MULTI_STRUCT_ERR	134	Failure to confirm integrity or repair data logging device mult-structure information
TKDATALOG_READ_ERR	135	Data logging device read failure
TKDATALOG_WRITE_ERR	136	Data logging device write failure
TKDATALOG_RECORDIDENTIFIER_ERR	137	Data logging device record identifier invalid
TKDATALOG_INSUFFMEM_ERR	138	Data logging device insufficient memory
TKDATALOG_CARD_CORRUPT_ERR	139	Logcard data logging device corrupted
TKDISP_INVALID_CHAR_ERR	140	Display character provided by screen content or host PC is out of range
TKDISP_INVALIDPARAM_ERR	141	Display coordinate provided by screen content or host PC is out of range
TKDISP_STRING_TOO_LONG_ERR	142	Display string and coordinates provided by screen content or host PC is exceeds display capability
TKDISP_STRING_TOO_HIGH_ERR	143	Display string and coordinates provided by screen content or host PC exceeds display capability
TKDISP_INVALID_LANG_ERR	144	Display language provided by screen content or host PC is out of range
TKEEPROM_INIT_ERR	150	EEPROM toolkit functions not initialized
TKEEPROM_ACK_ERR	151	Failure to receive ACK from EEPROM during read or write operation
TKEEPROM_STOP_ERR	152	Failure to receive ACK from EEPROM during stop operation
TKEEPROM_INVALID_END_ADDR	153	EEPROM address exceeds size of device
TKEEPROM_WRITE_TIMEOUT_ERR	154	<Not Used>
TKEEPROM_WRITE_READ_ERR	155	Timeout waiting for EEPROM write cycle to complete
TKEEPROM_WRITE_VERIFY_ERR	156	Failure to verify data written to EEPROM after write cycle has completed
TKEEPROM_CHKSM_READ_ERR	157	Failure to read EEPROM during checksum computation
TKFRAME_CSAFE_FRAME_STUFF_ERR	160	Failure in CSAFE frame processing during byte-unstuffing
TKFRAME_CSAFE_FRAME_CHKSM_ERR	161	CSAFE frame checksum failure
TKFRAME_NO_SCI_FRAME_ERR	162	No complete CSAFE frame detected during serial interface character processing

Internal Name	Value	Description
TKFRAME_NO_USB_FRAME_ERR	163	No complete CSAFE frame detected during USB interface character processing
TKFRAME_CSAFE_INVALID_SHORT_CMD_ERR	164	Invalid short command present in CSAFE frame
TKFRAME_CSAFE_INVALID_LONG_CMD_ERR	165	Invalid long command present in CSAFE frame
TKFRAME_CSAFE_FRAME_TOO_LONG_ERR	166	CSAFE frame exceeds maximum allowable frame length
TKFRAME_NO_EXPRF_FRAME_ERR	167	No complete CSAFE frame detected during 802.11 interface character processing
TKFRAME_CSAFE_INVALID_LONG_RSP_ERR	168	Invalid CSAFE frame length in slave response
TKHDW_EVENT_BURST_STACK_OVF_ERR	170	Burst stack overflow error
TKHDW_EVENT_BURST_STACK_UNF_ERR	171	Burst stack underflow error
TKHRTMON_INVALID_NUM_MEAS_ERR	180	Number of heartrate monitor measurements requests exceeds maximum
TKHRTMON_TOO_FEW_MEAS_ERR	181	Number of available heartrate monitor measurements is fewer than the requested number of measurements
TKMEM_INVALID_MEMTYPE_ERR	200	Requested memory operation specified invalid memory type
TKMEM_INVALID_START_ADDR_ERR	201	Requested memory operation specified invalid start address for the memory type
TKMEM_INVALID_END_ADDR_ERR	202	Requested memory operation specified invalid end address for the memory type
TKMEM_FLASH_WRITE_ERR	203	Flash memory write failure
TKRTTIMER_INVALID_MONTH_ERR	210	Invalid month specified in date structure during date format conversion
TKRTTIMER_INVALID_DAY_ERR	211	Invalid day specified in date structure during date format conversion
TKRTTIMER_INVALID_TIMER_NUM_ERR	212	Invalid task timer number specified during timer configuration
TKRTTIMER_INVALID_TIMER_MODE_ERR	213	Invalid task timer mode specified during timer configuration
TKSCI_INVALID_PORT_ERR	220	Invalid serial communication interface port specified
TKSCI_TX_SEND_ERR	221	Failure during serial communication interface transmission
TKSCI_RX_TIMEOUT_ERR	222	Receive timeout on serial communication interface
TKSCRN_INVALID_SPECFUNCTYPE	230	Special function type provided by the screen content or host PC is out of range
TKSCRN_ILLEGAL_SPLITDURATION	231	Illegal split duration was detected and fixed
TKSMCD_ACK_ERR	240	Failure to receive ACK from smart card during read operation
TKSMCD_STOP_ERR	241	Failure to receive ACK from smart card during stop operation
TKSMCD_INVALID_END_ADDR	242	Smart card address exceeds size of device
TKSMCD_WRITE_TIMEOUT_ERR	243	<Not Used>

Internal Name	Value	Description
TKSMCD_WRITE_READ_ERR	244	Timeout waiting for smart card write cycle to complete
TKSMCD_WRITE_VERIFY_ERR	245	Failure to verify data written to smart card after write cycle has completed
TKSMCD_CHKSM_READ_ERR	246	Failure to read smart card during checksum computation
TKSMCD_ACK_ERR_WRITE	247	Failure to receive ACK from smart card during write operation
TKTACH_INVALID_NUM_MEAS_ERR	250	Number of flywheel tach measurements requests exceeds maximum
TKTACH_TOO_FEW_MEAS_ERR	251	Number of available flywheel tach measurements is fewer than the requested number of measurements
TKTIME_INVALID_MONTH_ERR	260	Invalid month specified in date structure during date format conversion
TKTIME_INVALID_DAY_ERR	261	Invalid day specified in date structure during date format conversion
TKUSER_INIT_ERR	260	<Not Used>
TKCRC_ERR	300	<Not Used>
TKUSB_BAD_DESC_RQT_ERR	330	USB communication interface bad descriptor request
TKUSB_INVALID_EPNUM_ERR	331	Invalid USB communication interface endpoint specified
TKUSB_RX_TIMEOUT_ERR	332	<Not Used>
TKUSB_EPNUM_RX_OVERRUN	333	USB communication interface endpoint receive overrun
TKUSB_INIT_EPNUM_ERR	334	USB communication interface endpoint initialization failure
TKUSB_GET_RX_CHAR_ERR	335	Failure to get character from the USB communication interface endpoint
TKUSB_BUS_DISABLE_ERR	336	<Not Used>
TKUSB_BUS_RESET_ERR	337	<Not Used>
TKUSB_NO_FEATURE_REPORT_ERR	338	No feature report available on the USB communication interface
TKUSB_INVALID_STRING_ID_ERR	339	Invalid string descriptor ID request on the USB communication interface
TKUSB_EP_TX_OVERRUN_ERR	340	USB communication interface endpoint transmit overrun
TKUSB_INVALID_TX_LEN_ERR	341	Length of transmit data for USB communication interface exceeds buffer size
TKDIAG_DIAGFAIL_ERR	500	One or more diagnostic tests failed
TKDIAG_FLSHFONTDIAG_BADHDRCRC_ERR	501	Flash font information header CRC failure
TKDIAG_FLSHFONTDIAG_CRCCALC_ERR	502	Flash font information CRC calculation unsuccessful
TKDIAG_FLSHFONTDIAG_BADFONTCRC_ERR	503	Flash font information CRC failure
TKDIAG_FLSHSCRNDIAG_BADHDRCRC_ERR	510	Flash screen content information header CRC failure
TKDIAG_FLSHSCRNDIAG_CRCCALC_ERR	511	Flash screen content information CRC calculation unsuccessful
TKDIAG_FLSHSCRNDIAG_BADSCRNCRC_ERR	512	Flash screen content information CRC failure

Internal Name	Value	Description
TKDIAG_FLSHAPPDIAG_BADHDRCRC_ERR	520	Flash application information header CRC failure
TKDIAG_FLSHAPPDIAG_CRCCALC_ERR	521	Flash application information CRC calculation unsuccessful
TKDIAG_FLSHAPPDIAG_BADAPPCRC_ERR	522	Flash application information CRC failure
TKDIAG_UARTDIAG_UART1_INIT_ERR	530	Serial communication UART1 initialization failure
TKDIAG_UARTDIAG_UART1_WRITE_ERR	531	Serial communication UART1 loopback write failure
TKDIAG_UARTDIAG_UART1_READ_ERR	532	Serial communication UART1 loopback read failure
TKDIAG_UARTDIAG_UART2_INIT_ERR	533	Serial communication UART2 initialization failure
TKDIAG_UARTDIAG_UART2_WRITE_ERR	534	Serial communication UART2 loopback write failure
TKDIAG_UARTDIAG_UART2_READ_ERR	535	Serial communication UART2 loopback read failure
TKDIAG_ADCONVDIAG_INIT_ERR	540	Analog-to-digital converter initialization failure
TKDIAG_ADCONVDIAG_NOTREADY_ERR	541	<Not Used>
TKDIAG_ADCONVDIAG_ADCINPUT_ERR	542	Analog-to-digital converter conversion out of range failure
TKDIAG_SWUSERCONFIRM_ERR	550	User switch confirmation timeout failure
TKDIAG_SWSHORT_ERR	551	User switch short failure
TKDIAG_SW0_ERR	552	User switch 0 assertion failure
TKDIAG_SW1_ERR	553	User switch 1 assertion failure
TKDIAG_SW2_ERR	554	User switch 2 assertion failure
TKDIAG_SW3_ERR	555	User switch 3 assertion failure
TKDIAG_SW4_ERR	556	User switch 4 assertion failure
TKDIAG_SW5_ERR	557	User switch 5 assertion failure
TKDIAG_SW6_ERR	558	User switch 6 assertion failure
TKDIAG_SW7_ERR	559	User switch 7 assertion failure
TKDIAG_AMUXDIAG_NOTREADY_ERR	560	Analog mux analog-to-digital conversion not available failure
TKDIAG_AMUXDIAG_ANALOGVREFCHAN_ERR	561	Analog mux reference channel conversion out of range failure
TKDIAG_AMUXDIAG_ANALOGGNDCHAN_ERR	562	Analog mux ground channel conversion out of range failure
TKDIAG_VSUPPLYDIAG_VEXPDIAG_ERR	570	Expansion module voltage supply out of range failure
TKDIAG_VSUPPLYDIAG_GENINDIAG_ERR	571	Flywheel generator input voltage out of range failure
TKDIAG_VSUPPLYDIAG_VBATEXPDIAG_ERR	572	Battery expansion module voltage out of range failure
TKDIAG_VSUPPLYDIAG_VBATPROTDIAG_ERR	573	Battery protected voltage out of range failure
TKDIAG_VSUPPLYDIAG_VUSBDIAG_ERR	574	USB input voltage out of range failure
TKDIAG_VSUPPLYDIAG_VREFDIAG_ERR	575	Reference voltage out of range failure
TKDIAG_VSUPPLYDIAG_VBIASDIAG_ERR	576	LCD bias voltage supply out of range failure
TKDIAG_EXTEEDIAG_RDDATA1_ERR	580	EEPROM read data location 1 failure
TKDIAG_EXTEEDIAG_INVALIDCRC1_ERR	581	EEPROM manufacturing structure CRC validity failure

Internal Name	Value	Description
TKDIAG_EXTEEDIAG_RDDATA2_ERR	582	EEPROM read data location 2 failure
TKDIAG_EXTEEDIAG_INVALIDCRC2_ERR	583	EEPROM calibration structure CRC validity failure
TKDIAG_EXTEEDIAG_WRDATA1_ERR	584	EEPROM write data location 1 failure
TKDIAG_EXTEEDIAG_WRDATA2_ERR	585	EEPROM write data location 2 failure
TKDIAG_EXTEEDIAG_DATA1_ERR	586	EEPROM write/read verify data location 1 failure
TKDIAG_EXTEEDIAG_DATA2_ERR	587	EEPROM write/read verify data location 2 failure
TKDIAG_TACHDIAG_USERCONFIRM_ERR	590	Tach confirmation timeout failure
TKDIAG_TACHDIAG_TACHUNPLUG_ERR	591	<Not Used>
TKDIAG_TACHDIAG_TACHPLUG_ERR	592	Tach input is active or flywheel spinning failure
TKDIAG_TACHDIAG_TACHSPINNING_ERR	593	Tach input flywheel not spinning failure
TKDIAG_TACHDIAG_USERABORT_ERR	594	Tach user diagnostic abort failure
TKDIAG_HRTMONDIAG_USERCONFIRM_ERR	600	Heartrate montor confirmation timeout failure
TKDIAG_HRTMONDIAG_HRTUNPLUG_ERR	601	<Not Used>
TKDIAG_HRTMONDIAG_HRTPLUG_ERR	602	Heartrate monitor is active or pulses present failure
TKDIAG_HRTMONDIAG_HRTACTIVE_ERR	603	Heartrate monitor no pulses present failure
TKDIAG_HRTMONDIAG_USERABORT_ERR	604	Heartrate monitor user diagnostic abort failure
TKDIAG_GENINPUTDIAG_USERCONFIRM_ERR	610	Generator confirmation timeout failure
TKDIAG_GENINPUTDIAG_THRESHMAX_ERR	611	Generator voltage above max threshold failure
TKDIAG_GENINPUTDIAG_THRESHMIN_ERR	612	Generator voltage below min threshold failure
TKDIAG_GENINPUTDIAG_USERABORT_ERR	613	Generator user diagnostic abort failure
TKDIAG_SCDIAG_USERCONFIRM_ERR	620	Smart card confirmation timeout failure
TKDIAG_SCDIAG_ILLEGALDETECT_ERR	621	Smart card detect if no card present failure
TKDIAG_SCDIAG_DETECT_ERR	622	Smart card no detect if card present failure
TKDIAG_SCDIAG_COMM_ERR	623	Smart card communication failure
TKDIAG_SCDIAG_USERABORT_ERR	624	Smart card user diagnostic abort failure
TKDIAG_EXPCFREG_NOTPRESENT_ERR	660	Expansion configuration register module not present failure
TKDIAG_EXPCFREG_LO_ERR	661	Expansion configuration register low loopback signal failure
TKDIAG_EXPCFREG_HI_ERR	662	Expansion configuration register high loopback signal failure
TKDIAG_EXPSTSLED_NOTPRESENT_ERR	670	Expansion status LED module not present failure
TKDIAG_EXPFLASH_NOTPRESENT_ERR	680	Expansion Flash memory module not present failure
TKDIAG_EXPFLASH_FILLNORMALDATA_ERR	681	Expansion Flash memory fill normal data failure
TKDIAG_EXPFLASH_NORMALDATA_ERR	682	Expansion Flash memory normal data verify failure
TKDIAG_EXPFLASH_FILLINVERTEDDATA_ERR	683	Expansion Flash memory fill inverted data failure
TKDIAG_EXPFLASH_INVERTEDDATA_ERR	684	Expansion Flash memory inverted data verify failure

Internal Name	Value	Description
TKDIAG_EXPSRAM_NOTPRESENT_ERR	690	Expansion static RAM module not present failure
TKDIAG_EXPSRAM_NORMALDATA_ERR	691	Expansion static RAM normal data write/read verify failure
TKDIAG_EXPSRAM_INVERTEDDATA_ERR	692	Expansion static RAM inverted data write/read verify failure
TKDIAG_EXPEEDIAG_NOTPRESENT_ERR	700	Expansion EEPROM module not present failure
TKDIAG_EXPEEDIAG_INVALIDCRC_ERR	701	Expansion EEPROM manufacturing information CRC valid check failure
TKDIAG_EXPEEDIAG_RDDATA1_ERR	702	Expansion EEPROM read failure
TKDIAG_EXPEEDIAG_WRDATA1_ERR	703	Expansion EEPROM write failure
TKDIAG_EXPEEDIAG_DATA1_ERR	704	Expansion EEPROM write/read verify failure
TKDIAG_EXP232DIAG_NOTPRESENT_ERR	710	Expansion RS232 interface module not present failure
TKDIAG_EXP232DIAG_CONFIG_ERR	711	Expansion RS232 configuration failure
TKDIAG_EXP232DIAG_TXCHAR_ERR	712	Expansion RS232 loopback transmit failure
TKDIAG_EXP232DIAG_LOOPBACK_TO_ERR	713	Expansion RS232 loopback receive timeout failure
TKDIAG_EXP232DIAG_LOOPBACK_DATA_ERR	714	Expansion RS232 loopback data verify failure
TKDIAG_EXP232DIAG_USERCONFIRM_ERR	715	Expansion RS232 confirmation timeout failure
TKDIAG_EXP232DIAGDIAG_USERABORT_ERR	716	Expansion RS232 user diagnostic abort failure
TKDIAG_EXP485DIAG_NOTPRESENT_ERR	720	Expansion RS485 interface module not present failure
TKDIAG_EXP485DIAG_CONFIG_ERR	721	Expansion RS485 configuration failure
TKDIAG_EXP485DIAG_FORMATFRAME_ERR	722	Expansion RS485 CSAFE frame format failure
TKDIAG_EXP485DIAG_SENDCMD_ERR	723	Expansion RS485 CSAFE frame send failure
TKDIAG_EXP485DIAG_UNFORMATFRAME_ERR	724	Expansion RS485 CSAFE frame unformat failure
TKDIAG_EXP485DIAG_USERCONFIRM_ERR	725	Expansion RS485 confirmation timeout failure
TKDIAG_EXP485DIAGDIAG_USERABORT_ERR	726	Expansion RS485 user diagnostic abort failure
TKDIAG_EXP485DIAG_NORESPONSE_ERR	727	Expansion RS485 no command response failure
TKDIAG_EXPWIFIDIAG_NOTPRESENT_ERR	730	Expansion 802.11 module not present failure
TKDIAG_EXPWIFIDIAG_CFINIT_ERR	731	Expansion 802.11 initialization failure
TKDIAG_EXPWIFIDIAG_RECONFIG_ERR	732	Expansion 802.11 task reconfiguration failure
TKDIAG_EXPWIFIDIAG_USERABORT_ERR	733	Expansion 802.11 user diagnostic abort failure
TKDIAG_EXPWIFIDIAG_DHCP_TO_ERR	734	Expansion 802.11 DHCP timeout failure
TKDIAG_EXPWIFIDIAG_CFNOTPRESENT_ERR	735	Expansion 802.11 correct CF card present failure
TKEXP_RS232_INVALID_ERR	1000	<Not Used>
TKEXP_CF_NOTPRESENT_ERR	1001	Expansion module CF card not present
TKEXP_CF_CIRQINVALID_ERR	1002	Expansion module CF IRQ invalid state
TKEXP_CF_CARDNOTREADY_ERR	1003	Expansion module CF card not ready

Internal Name	Value	Description
TKEXP_CF_MEMTEST_ERR	1004	Expansion module memory test failure
TKEXP_CF_INVALIDSTATE_ERR	1005	Expansion module invalid state
TKEXP_CF_RFVENDORSTRING_ERR	1006	Expansion module invalid 802.11 vendor string
TKEXP_INVALIDLEDMODE_ERR	1007	Expansion module invalid status LED mode
TKEXP_INVALIDLEDCOLOR_ERR	1008	Expansion module invalid status LED color
TKDEBUG_INIT_ERR	2000	<Not Used>
IOADCONV_BG_TIMEOUT_ERR	810	Analog-to-digital converter bandgap reference ready timeout
IOADCONV_RESET_TIMEOUT_ERR	811	Analog-to-digital converter reset timeout
IOADCONV_INVALID_CHAN_ERR	812	Analog-to-digital converter invalid channel
IOADCONV_NOT_RDY_ERR	813	Analog-to-digital converter not ready
IOADCONV_INVALID_REF_ERR	814	Analog-to-digital converter invalid reference level
IOADCONV_INIT_ADC_ERR	815	<Not Used>
IODMA_INVALID_MEM_CHAN_ERR	820	DMA invalid memory channel requested
IODMA_INVALID_IO_RQST_CHAN_ERR	821	DMA invalid IO channel requested
IODMA_INIT_DMA_ERR	822	DMA initialization unsuccessful
IODMA_QUEUE_FULL_ERR	823	DMA queue full
IOHDW_MEM_INVALID_CS_ERR	830	Memory chip select invalid
IOHDW_INVALID_DMACLK_ERR	831	DMA clock invalid
IOHDW_INVALID_SYSCLK_ERR	832	System clock invalid
IOI2C_NOACK_ERR	840	I2C no ACK on read or write
IOI2C_INIT_WDR_TIMOUT_ERR	841	I2C initialization wait for data ready timeout
IOI2C_INIT_XMIT_TIMOUT_ERR	842	I2C initialize transmit data timeout
IOI2C_SEND_XMIT_TIMOUT_ERR	843	I2C send transmit data timeout
IOI2C_GET_RECV_TIMOUT_ERR	844	I2C get receive data timeout
IOI2C_STOP_TIMEOUT_ERR	845	I2C stop condition timeout
IOI2C_WDR_TIMOUT_ERR	846	I2C wait for data ready timeout
IOI2C_INVALID_BAUD	847	I2C invalid baud rate
IOLCD_DISPINIT_ERR	860	LCD display initialization unsuccessful
IOLCD_INVALIDPARAM_ERR	861	LCD invalid display parameter specified
IOMEM_FLASH_ERASE_TIMEOUT_ERR	870	Flash memory erase timeout
IOMEM_FLASH_WRITE_TIMEOUT_ERR	871	Flash memory write timeout
IORTCLOCK_WRITE_TIME_ERR	880	Real-time clock write timeout
IOSCI_INVALID_PORT_ERR	890	Serial communication interface invalid port
IOSCI_INVALID_BAUD_ERR	891	Serial communication interface invalid baud

Internal Name	Value	Description
IOSCI_INVALID_CNT_ERR	892	Serial communication interface characters requested out of range
IOSCI_INIT_PORT_ERR	893	Serial communication interface unsuccessful port initialization
IOSCI_TXOVERRUN_ERR	894	Serial communication interface transmit overrun
IOSCI_RXOVERRUN_ERR	895	Serial communication interface receive overrun
IOSCI_RXFRAME_ERR	896	Serial communication interface framing failure
IOSCI_RXPARITY_ERR	897	Serial communication interface parity failure
IOSCI_RXBREAK_ERR	898	Serial communication interface break condition failure
IOTIMER_INVALID_TIMERID_ERR	910	Timer ID invalid
IOUSER_SEMAPHORE_PEND_ERR	920	<Not Used>
IOUSER_SEMAPHORE_POST_ERR	921	<Not Used>
IOUSB_RST_TIMOUT_ERR	930	USB interface reset timeout
IOUSB_CFG_TIMOUT_ERR	931	USB interface configuration timeout
IOUSB_CFG_ENDPT_ERR	932	USB interface endpoint configuration timeout
IOUSB_SETUP_ERR	933	USB interface control endpoint packet setup failure
IOUSB_FIFO_RD_ERR	934	USB interface FIFO read failure
IOUSB_NULL_PTR_ERR	935	USB interface null pointer
IOUSB_BUS_INIT_ERR	936	USB interface bus initialization failure
IOUSB_TX_BUFFER_ERR	937	USB interface null Tx buffer pointer
IOUSB_EP_BUSY_ERR	938	USB interface endpoint busy
IOUSB_EP_INVALID_ERR	939	USB interface invalid endpoint specified
IOUSB_WAKEUP_DISABLE_ERR	940	USB interface bus unable to resume
IOUSB_BAD_FRAMENUM_ERR	941	USB interface bad frame number
IOUSB_CFG_DEV_ERR	942	USB interface not configured
IOUSB_BAD_IFCNUM_ERR	943	USB interface invalid interface number


concept 2
ROWING