OCPI 2.3.0

Open Charge Point Interface

https://github.com/ocpi & https://evroaming.org/

document version 2.3.0, 2025-02-21

Table of Contents

EVRoaming Foundation

OCPI is developed and managed by the EVRoaming Foundation. The EVRoaming Foundation is a contributor based organisation. Everyone can join the EVRoaming Foundation via https://www.evroaming.org

The EVRoaming Foundation aims to keep OCPI as free from IPR as possible. If you want to contribute by adding new functionality or features, you are required to send us the signed Contributor Agreement (CA) document before contributing. To get the CA, ask for it by sending an email to: info@evroaming.org.

Version History

Version	Date	Author	Description
2.3.0-rc2	2025-01-16	Greg Fitzpatrick *ChargeHub* Petar Jovevski *Metergram* Robert Gliguroski *Metergram* Philipp Fischbacher *ChargePoint* Reinier Lamers *SWTCH Energy*	• Make OCPI extensible • Add vehicle types to EVSE • Add list of accepted eMSPs to EVSE • Add a Parking object linked to EVSE • Information for people with disabilities • Support for North American taxes • 15118 Plug and Charge compatibility flags on Connectors • Make Hub support incremental from regular multi-party Platform support
2.2.1-d2	2023-09-07	Jakub Karbownik *Ekoenergetyka* Rudolph Froger *TandemDrive* Robert de Leeuw *EVA Global* Reinier Lamers *ihomer*	Documentation update.
2.2.1	2021-10-06	Robert de Leeuw *ihomer* Reinier Lamers *ihomer*	final release of OCPI 2.2.1. Added country_code and party_id to CdrToken class. Fixed datatype of CDR SignedData URL. Improved some descriptions. Fixed length of CDR SignedData, increased to 5000. Change signed data related fields to string postal_code optional in CdrLocation. state added to CdrLocation. AC_2_PHASE and AC_2_PHASE_SPLIT added to PowerType in Connector Additional types added to ConnectorType in Connector Added `connector_id` to StartSession command and `START_SESSION_CONNECTOR_REQUIRED` to EVSE Capabilities. Added optional field: `home_charging_compensation` to CDR. Improved description.examples Tariff for `step_size`
2.2-d2	2020-06-12	Robert de Leeuw *ihomer*	2nd documentation revision of the OCPI 2.2 spec. Contains textual improvements and fixes some of the examples. Most improvements in the tarrifs module, especially `step_size` is better explained.

Version	Date	Author	Description
2.2	2019-09-30	Robert de Leeuw *ihomer*	Added support for Roaming Hubs Adds support for Platforms with multiple/different roles, additional roles Adds support for smart charging Lots of improvements to existing modules See changelog
2.1.1-d2	2019-06-21	Robert de Leeuw ihomer	Fixes the command module documentation, fixes a lot of examples, lots of small textual improvements: see changelog
2.1.1	2017-06-08	Robert de Leeuw *ihomer*	Fixed 4 bugs found in OCPI 2.1, lots of small textual improvements: see changelog
2.1	2016-04-08	Robert de Leeuw *ihomer*	Added command module. Added support for real-time authorization. Lots of small improvements: see changelog
2.0-d2	2016-02-15	Robert de Leeuw *ihomer*	2nd documentation revision of the OCPI 2.0 spec. Only documentation updated: ConnectorType of Connector was not visible, credentials clarified, location URL segments incorrect (now string, was int), minor textual updates. DateTime with timezones is still an issue
2.0	2015-12-30	Robert de Leeuw *ihomer* Simon Philips *Becharged* Chris Zwirello *The New Motion* Simon Schilling	First official release of OCPI.
0.4	2014-11-04	Olger Warnier *The New Motion*	First draft of OCPI. (Also known as Draft v4)
0.3	2014-05-06	Olger Warnier *The New Motion*	First draft of OCPI. (Also known as Draft v3)

Document revisions There can be multiple documentation revisions of the same version of the OCPI protocol.

The newer documentation revisions of the same protocol version can never change the content of the messages: no new fields or renaming of fields. A new revision can only clarify/fix texts/descriptions and fix typos etc.

These documentation revisions (not the first) will be named: d2, d3, d4 etc.

Examples:

- OCPI 2.1.1 is a different protocol version of OCPI than OCPI 2.1.
- OCPI 2.2-d2 is the same protocol version as OCPI 2.2, but a newer documentation revision.

1. OCPI

1.1. OCPI 2.3.0

OCPI 2.3.0 is a release with only the minimal changes to 2.2.1 that are necessary to meet two requirements: first, to comply with new laws and regulations coming into effect in 2025, and second, to provide room for custom extensions to OCPI without breaking compatibility without other implementers.

The changes that are made to address these requirements are:

- make it possible to define extra modules, fields, enumeration values for certain enums,
- add a Parking object to give information about the parking at an EVSE, and whether it is suitable for heavy-duty vehicles or people with disabilities,
- support EVSE information for people with disabilities,
- support North American tax structures,
- include some straightforward enumeration values from the OCPI 3.0 draft, including the ones indicating ISO 15118 support, and
- add a new field in the Credentials object to give a hub's party ID, which allows platforms with hub support to connect to platforms that don't implement hub functionality.
- add new Payments module

1.2. OCPI 2.2.1

During implementation of OCPI 2.2 some issues where found that required updating the protocol to fix them. These are all minor changes, so most OCPI 2.2 implementations would need no, or only minor changes, to upgrade to OCPI 2.2.1.

For more information on detailed changes see changelog.

1.3. OCPI 2.2

OCPI 2.2 includes new functionality and improvements, compared to OCPI 2.1.1.

1.3.1. Changes/New functionality:

- Support for Hubs
  - Message routing headers
  - Hub Client Info
- Support Platforms with multiple/different roles, additional roles
- Charging Profiles
- Preference based Smart Charging
- Improvements:
  - CDRs: Credit CDRs, VAT, Calibration law/Eichrecht support, Session_id, AuthorizationReference,

CdrLocation, CdrToken

- Sessions: VAT, CdrToken, How to add a Charging Period
- Tariffs: Tariff types, Min/Max price, reservation tariff, Much more examples
- Locations: Multiple Tariffs, Lots of small improvements
- Tokens: Group_id, energy contract
- Commands: Cancel Reservation added

For more information on detailed changes see changelog.

1.4. Introduction and background

The Open Charge Point Interface (OCPI) enables a scalable, automated EV roaming setup between Charge Point Operators and e-Mobility Service Providers. It supports authorization, charge point information exchange (including live status updates and transaction events), charge detail record exchange, remote charge point commands and the exchange of smart-charging related information between parties.

It offers market participants in EV an attractive and scalable solution for (international) roaming between networks, avoiding the costs and innovation-limiting complexities involved with today's non-automated solutions or with central roaming hubs. As such it helps to enable EV drivers to charge everywhere in a fully-informed way, helps the market to develop quickly and helps market players to execute their business models in the best way.

What does it offer (main functionality):

- A good roaming system (for bilateral usage and/or via a Roaming Hub).
- Real-time information about location, availability and price.
- A uniform way of exchanging data (Notification Data Records and Charge Data Records), before during and after the transaction.
- Remote mobile support to access any Charge Point without pre-registration.

This document describes a combined set of standards based on the work done in the past. Next to that, the evolution of these standards and their use are taken into account and some elements have been updated to match current use.

OCPI is developed with support of:

evRoaming4EU project and its partners:

ECISS project and its partners:

The latest version of this specification can be found here: https://github.com/ocpi/ocpi

2. Terminology and Definitions

2.1. Requirement Keywords

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in https://www.ietf.org/rfc/rfc2119.txt.

2.2. Abbreviations

Abbr.	Description
CDR	Charge Detail Record.
CPO	Charging Point Operator.
eMSP	e-Mobility Service Provider.
EV	Electric Vehicle.
EVSE	Electric Vehicle Supply Equipment. Is considered as an independently operated and managed part of a Charge Point that can deliver energy to one EV at a time.
JSON	JavaScript Object Notation.
NAP	National Access Point.
NSP	Navigation Service Provider.
OCPP	Open Charge Point Protocol.
SCSP	Smart Charging Service Provider.
PTP	Payment Terminal Provider.
PSP	Payment Service Provider.

2.3. EV Charging Market Roles

In the EV Charging landscape, different market roles can be identified.

Role	Description
CPO	Charging Point Operator. Operates a network of Charge Points.
eMSP	e-Mobility Service Provider. Gives EV drivers access to charging services.
NAP	National Access Point. Provides a national database with all (public) charging locations. Information can be sent and retrieved from the NAP. This makes it different from a typical NSP.
NSP	Navigation Service Provider. Provides EV drivers with location information of Charge Points. Usually only interested in Location information.

Role	Description
Roaming Hub	A business that facilitates roaming by offering integration with multiple roaming partners through one technical connection.
SCSP	Smart Charging Service Provider. Provides Smart Charging service to other parties. Might use a lot of different inputs to calculate Smart Charging Profiles.
PTP	Payment Terminal Provider. Refers to the party operating the payment terminal management system. This can be the terminal vendor (re/seller) and manufacturer
PSP	Payment Service Provider. Refers to the party providing acquiring services, which is typically a bank or institution that accepts and processes electronic payments. Merchant can enter into a direct contract with an acquirer or use the services of an intermediary (a payment service provider)

Some of these roles can be combined in one company. A Platform can provide service for multiple CPOs or eMSPs, but also for both eMSPs and CPOs.

OCPI 2.0 and OCPI 2.1.1 had a very strict definition of roles: only CPO and eMSP. But this is rare in the real world, there are almost no parties that are strictly CPO or eMSP and have their own platform. In the real world, lots of parties provide service to CPOs that are not running their own platform. A lot of CPOs are also eMSP. With OCPI 2.1.1 and earlier that meant having to set up an OCPI connection per role.

OCPI 2.2 introduced more roles and abstracts the role from the OCPI connection itself. OCPI 2.2, 2.2.1 and 2.3.0 are described in terms of about Platforms connecting to Platforms. The Platform itself is not a role. The Platform provides services for 1 or more roles.

Examples of platforms:

- A pure CPO: Not providing services to other CPOs. Not being an eMSP. Running its own software that connects via OCPI.
  Is defined in OCPI as a Platform has 1 CPO role, the CPO role of that company.
- A Company that has a cloud-based eMSP software solution, it offers to companies that want to be eMSP, but don’t want to host/run their own software.
  Is a Platform that has a number of eMSP roles, one for each eMSP the company is providing services for. Not for this company itself because the company itself is not an eMSP.
- A Company that operates public Charge Points and also provides eMSP service to EV drivers, running their own software platform.
  Is seen in OCPI as a Platform that has 2 roles: CPO and eMSP for this company.
- If one the companies above starts to offer their service to other CPOs and eMSP, it is in OCPI still seen as 1 platform. This platform then provides multiple CPO and eMSP roles.
- A Roaming Hub is in OCPI terms also a Platform. Other OCPI Platforms can connect to it.

2.3.1. Typical OCPI implementations per Role

The following table shows the typical modules implemented by the different roles. These are not required. The table shows the typical communication role: Receiver, Sender or Both.

Modules	CPO	eMSP	Roaming Hub	NSP	NAP	SCSP	PTP
CDRs	Sender	Receiver	Both				Receiver
Charging Profiles	Receiver		Both			Sender	
Commands	Receiver	Sender	Both				Sender
Credentials	Both	Both	Both	Both	Both	Both	Both
Hub Client Info	Receiver	Receiver	Sender	Receiver	Receiver	Receiver	
Locations	Sender	Receiver	Both	Receiver	Both		Receiver
Sessions	Sender	Receiver	Both			Receiver	Receiver
Tariffs	Sender	Receiver	Both	Receiver	Both		Receiver
Tokens	Receiver	Sender	Both				
Payments	Receiver						Sender
Versions	Both	Both	Both	Both	Both	Both	Both

2.4. Terminology

Term	Description
Broadcast Push	When communicating via a Hub, a data owner can do a single call to the Hub, the Hub then calls all receiving systems. See: Broadcast push
Charge Point	The physical system where an EV can be charged. A Charge Point has one or more EVSEs. Sometimes called Charging Station
Payment Terminal	A payment terminal allows a merchant to capture card information and to transmit this data to the acquiring party for authorization and finally to transfer funds to the merchant. In order to provide acquiring services on a payment terminal strict protocols and certifications apply.
Client Owned Objects	In a normal REST interface the server is the owner of data, when a new resource is created by calling POST, the server creates the URL where the resource can be found by a client. OCPI is different, in most modules the owner is the party pushing data to a server, to inform them of updates. For example Locations, the CPO owns a Location (Charge Point), when a new Charge Point is added, the CPO calls PUT on the eMSP systems to inform them about new locations. See: Client Owned Objects
Configuration Module	OCPI Module needed to setup and maintain OCPI connections, but does not provide information for the EV driver: Credentials, Versions and Hub Client Info. Configuration Modules do NOT use message routing.

Term	Description
Functional Module	OCPI Module that provides functionality/information for the EV Driver, such as: Tokens, Locations, CDRs etc. Functional Modules use message routing.
Hub	Functionality in an OCPI platform to route OCPI requests and responses based on their content.
Open Routing Request	This is for Platforms that are connected via a Hub. When a system sends a pull request to the Hub, and does not know, or care about, the owner of information, but asks the Hub to route the GET to the correct Platform. The Hub finds the correct Platform and routes the request to that Platform. See: Open Routing Request
Platform	Software that provides services via OCPI. A platform can provide service for a single eMSP or CPO, or for multiple CPOs or eMSPs. It can even provide services for both eMSPs and CPOs at the same time. A Platform will typically only provide services for a single Roaming Hub, through the Platform's Hub functionality.
Pull	A system calls GET request to retrieve information from the system that owns the data.
Push	The system (owning the data) actively calls POST/PUT/PATCH to update other systems with new/updated information.

2.5. Provider and Operator abbreviation

In OCPI it is advised to use eMI3/IDACS compliant names for Contract IDs and EVSE IDs. The provider and the operator name is important here, to target the right provider or operator, they need to be known upfront, at least between the cooperating parties.

In several standards, an issuing authority is mentioned that will keep a central registry of known Providers and Operators.

For more information about the format requirements for Contract IDs and EVSE IDs, and for authorities issuing Party IDs for providers or operators, see the EV Roaming Foundation's webpage on Contract and EVSE IDs: https://evroaming.org/contract-evse-ids/.

2.6. Charging topology

The charging topology, as relevant to the eMSP, consists of three entities:

- *Connector* is a specific socket or cable available for the EV to make use of.
- *EVSE* is the part that controls the power supply to a single EV in a single session. An EVSE may provide multiple connectors but only one of these can be active at the same time.
- *Location* is a group of one or more EVSEs that belong together geographically or spatially.

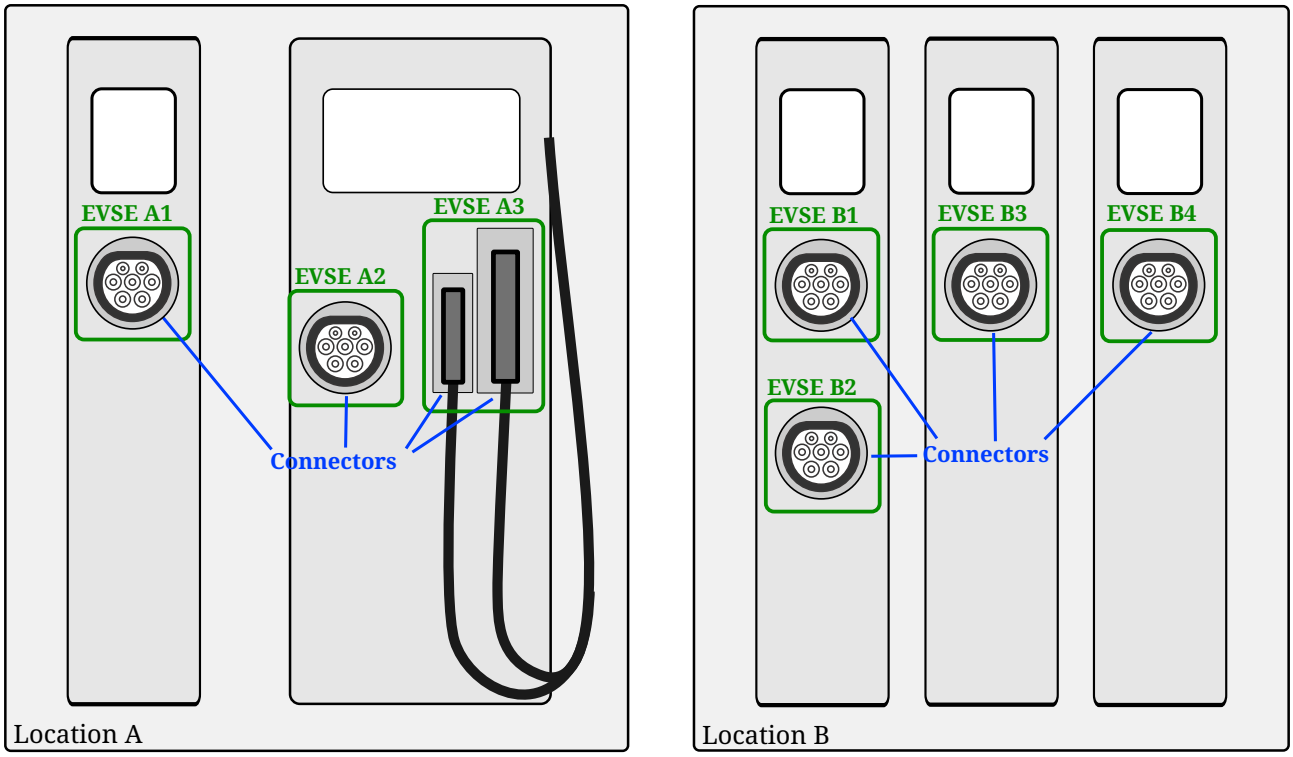


Figure 1. Charging Topology schematic

A Location is typically the exact location of one or more EVSEs, but it can also be the entrance of a parking garage or a gated community. It is up to the CPO to use whatever makes the most sense in a specific situation. Once arrived at the location, any further instructions to reach the EVSE from the Location are stored in the EVSE object itself (such as the floor number, visual identification or manual instructions).

2.7. Variable names

To prevent issues with capitals in variable names, the naming in JSON is not CamelCase but snake_case. All variables are lowercase and include an underscore for a whitespace.

2.8. Cardinality

When defining the cardinality of a field, the following symbols are used throughout this document:

Symbol	Description	Type
?	An optional object. If not set, it might be `null`, or the field might be omitted. When the field is set to null or omitted and it has a default value, the value is the default value.	Object
1	Required object.	Object
*	A list of zero or more objects. If empty, it might be `null`, `[]` or the field might be omitted.	[Object]
+	A list of at least one object.	[Object]

2.9. Data Retention

OCPI does not specify how long a system should store data. Companies are RECOMMENDED to make this part of

business contracts. Parties also will need to oblige to local legislation.

2.9.1. Between OCPI version

When a new version of OCPI is implemented, the data exchanged via the old version does not have to be available via the newer version of OCPI. Hence, the Version end-point will probably have different end-points per version. So when an object is stored with a URL that contains a version, it is NOT REQUIRED to be available at a URL with a different version number.

3. Supported Topologies

OCPI started as a bilateral protocol, for peer-to-peer communication. Soon parties started to use OCPI via Hubs, but OCPI 2.1.1 and earlier were not designed for that. OCPI 2.2 introduced a solution for this: message routing.

OCPI 2.2 introduced Platforms that connect via OCPI instead of CPO and eMSP, more on this in: EV Charging Market Roles

3.1. Peer-to-peer

The simplest topology is a bilateral connection: peer-to-peer between two platforms, and in the most simple version each platform only has 1 role.

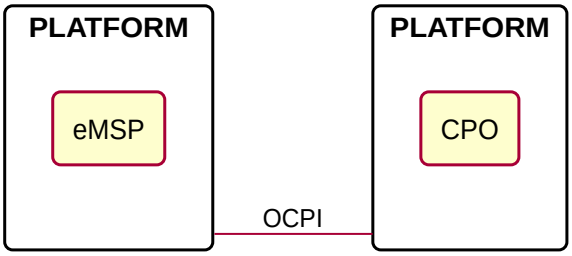


Figure 2. peer-to-peer topology example

3.2. Multiple peer-to-peer connections

A more real-world topology where multiple parties connect their platforms and each platform only has 1 role. (Not every party necessarily connects with all the other parties with the other role).

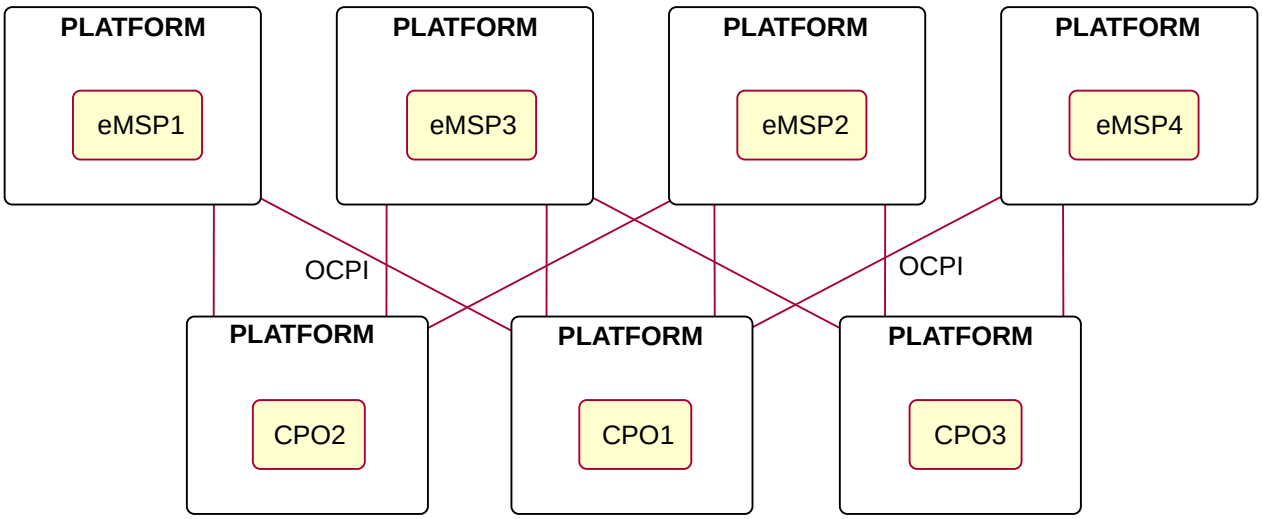


Figure 3. Multiple peer-to-peer topology example

3.3. Peer-to-peer multiple the same roles

Some parties provide for example CPO or eMSP services for other companies. So the platform hosts multiple parties with the same role. This topology is a bilateral connection: peer-to-peer between two platforms, and both platforms can have multiple roles.

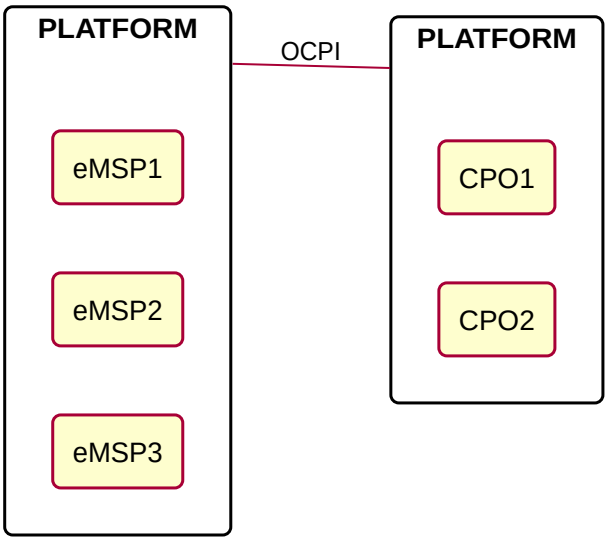


Figure 4. peer-to-peer with multiple roles topology example

3.4. Peer-to-peer dual roles

Some parties have dual roles, most of the companies are CPO and eMSP. This topology is a bilateral connection: peer-to-peer between two platforms, and both platforms have the CPO and the eMSP roles.

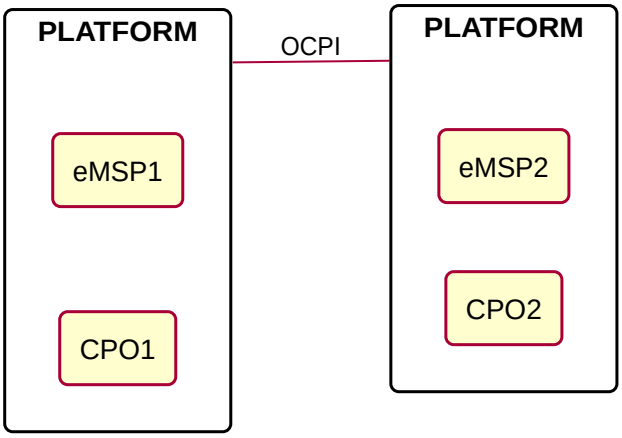


Figure 5. peer-to-peer with both CPO and eMSP roles topology example

3.5. Peer-to-peer mixed roles

Some parties have dual roles, or provide them to other parties and then connect to other companies that do the same. This topology is a bilateral connection: peer-to-peer between two platforms, and both platforms have multiple different and also the same roles.

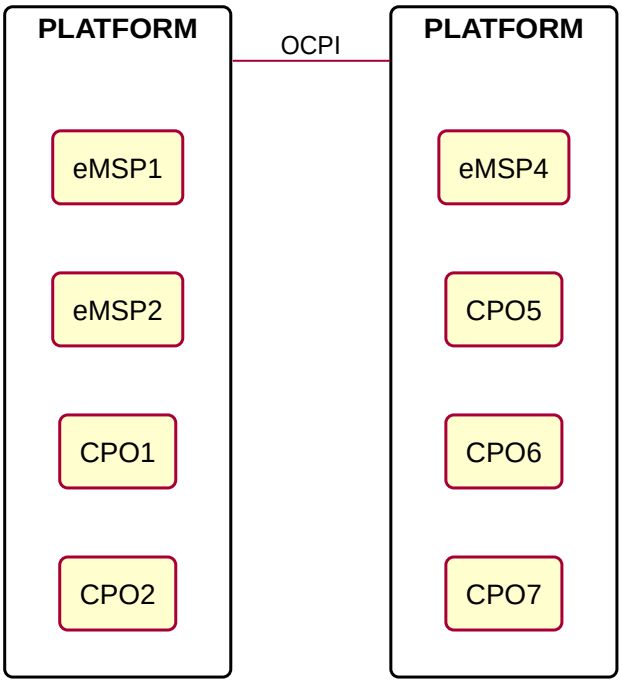


Figure 6. peer-to-peer with mixed roles topology example

3.6. Multiple peer-to-peer

More a real-world topology when OCPI is used between market parties without a hub, all parties are platforms with multiple roles.

Disadvantage of this: requires a lot of connections between platforms to be setup, tested and maintained.

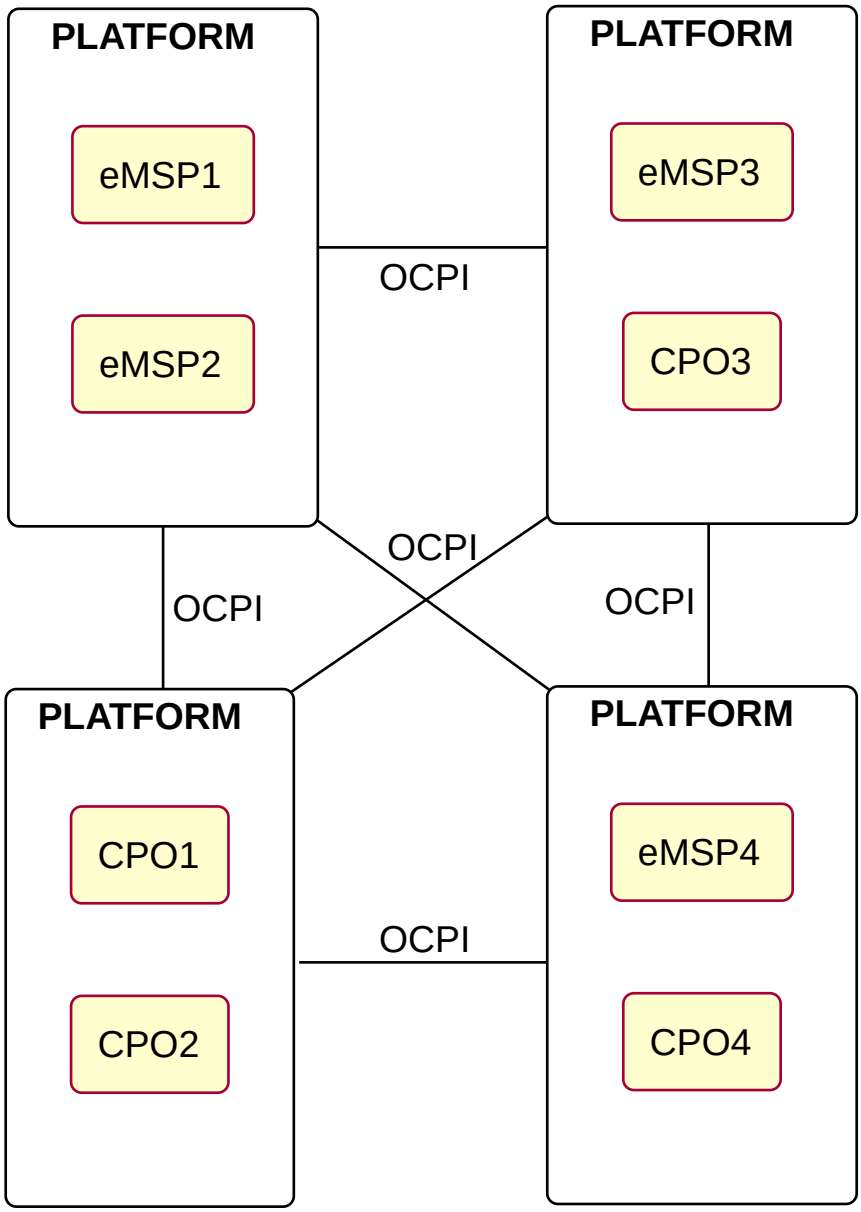


Figure 7. peer-to-peer with mixed roles topology example

3.7. Platforms via Hub

This topology has all Platforms only connect via a Hub, all communication goes via the Hub.

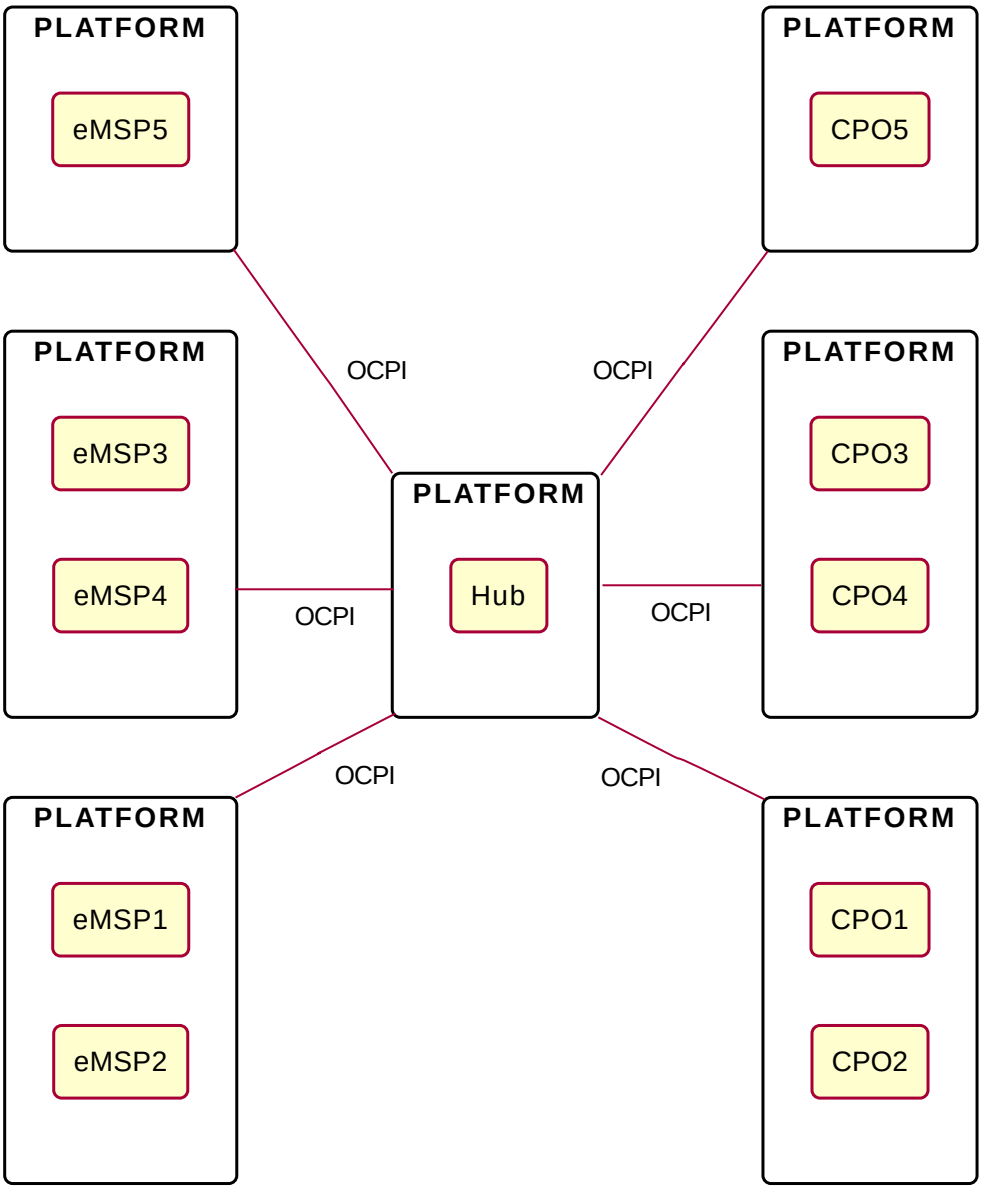


Figure 8. Platforms connected via a Hub topology example

3.8. Platforms via Hub and direct

Not all Platforms will only communicate via a Hub. There might be different reasons for Platforms to still have peer-to-peer connections. The Hub might not yet support new functionality. The Platforms use a custom module for some new project, which is not supported by the Hub. etc.

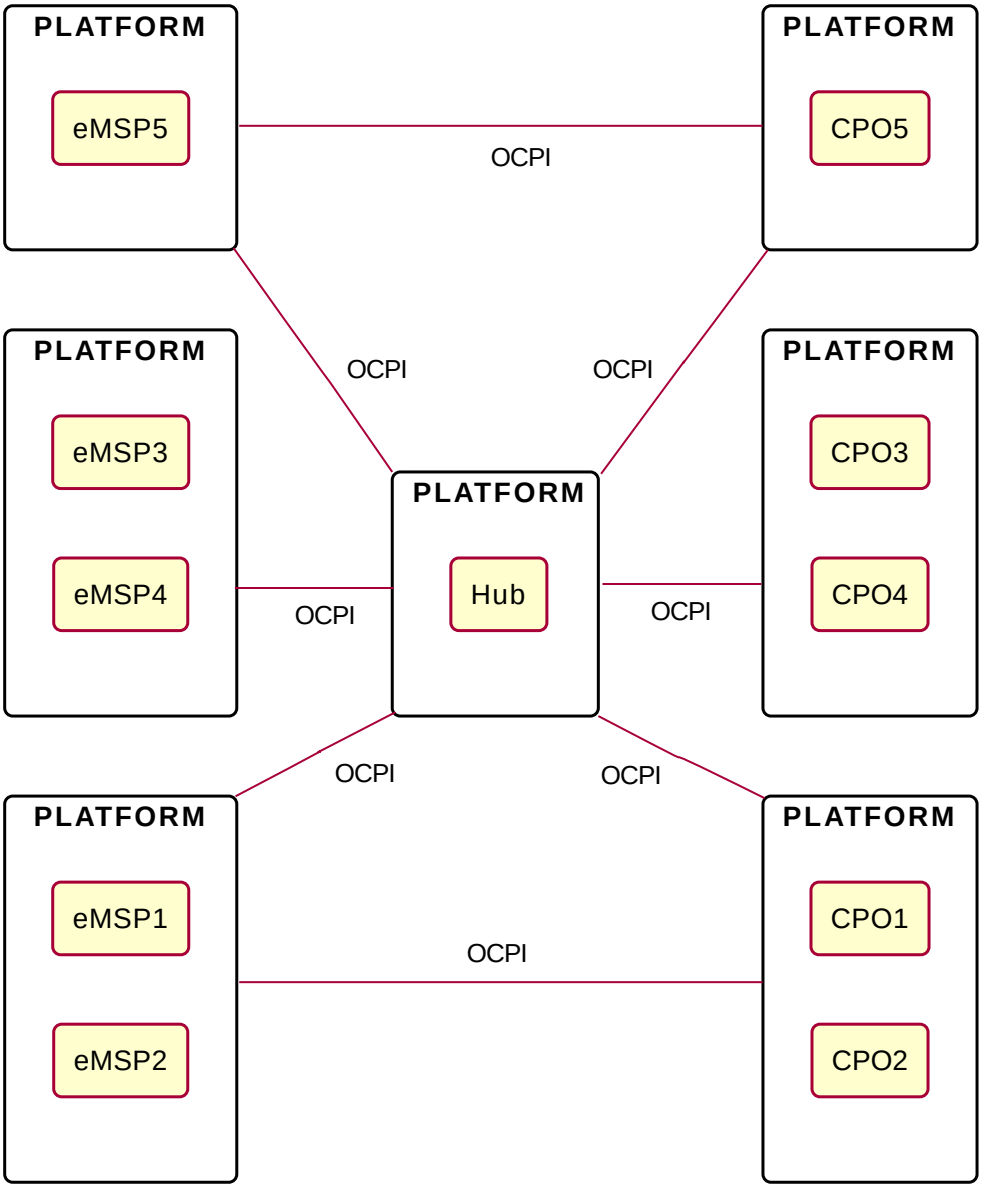


Figure 9. Platforms connected via a Hub and directly topology example

4. Transport and format

4.1. JSON / HTTP implementation guide

The OCPI protocol is based on HTTP and uses the JSON format. It follows a RESTful architecture for web services where possible.

4.1.1. Security and authentication

The interfaces are protected on the HTTP transport level, with SSL and token-based authentication. Please note that this mechanism does **not** require client-side certificates for authentication, only server-side certificates to set up a secure SSL connection.

4.1.2. Authorization header

Every OCPI HTTP request MUST add an 'Authorization' header. The header looks as follows:

```
Authorization: Token ZWJmM2IzOTktNzc5Zi00NDk3LTliOWQtYWM2YWQzY2M0NGQyCg==
```

NOTE	HTTP header names are case-insensitive

The literal 'Token' indicates that the token-based authentication mechanism is used, in OCPI this is called the 'credentials token'. 'Credentials tokens' are exchanged via the credentials module. These are different 'tokens' than the Tokens exchanged via the Token Module: Tokens used by drivers to authorize charging. To prevent confusion, when talking about the token used here in the HTTP Authorization header, call them: 'Credentials Tokens'.

After the literal 'Token', there SHALL be one space, followed by the 'encoded token'. The encoded token is obtained by encoding the credentials token to an octet sequence with UTF-8 and then encoding that octet sequence with Base64 according to RFC 4648.

So for example, to use the credentials token 'example-token' in an OCPI request, one should include this header:

```
Authorization: Token ZXhhbXBsZS10b2tlbgo=
```

NOTE	Many OCPI 2.1.1 and 2.2 implementations do not Base64 encode the credentials token when including it in the 'Authorization' header. Since OCPI 2.2-d2 the OCPI specification documents clearly require Base64 encoding the credentials token in the header value. Implementations that wish to be compatible with non-encoding 2.1.1 and 2.2 implementations have to choose the right way to parse and write authorization headers by either trial and error or configuration flags.

The credentials token must uniquely identify the requesting party. This way, the server can use the information in the Authorization header to link the request to the correct requesting party's account.

If the header is missing or the credentials token doesn't match any known party then the server SHALL respond with an HTTP `401 - Unauthorized` status code.

When a server receives a request with a valid `CREDENTIALS_TOKEN_A`, on another module than: `credentials` or

`versions`, the server SHALL respond with an HTTP `401 - Unauthorized` status code.

4.1.3. Pull and Push

OCPI supports both **Pull** and **Push** models.

- **Push:** Changes in objects and new objects are sent (semi) real-time to the receiver.
- **Pull:** Receiver request a (full) list of objects periodically.

OCPI doesn't require parties to implement Push. Pull is required, a receiver needs to be able to get *in-sync* after a period of connection loss.

It is possible to implement a Pull only OCPI implementation, it might be a good starting point for an OCPI implementation. However, it is strongly advised to implement Push for production systems that have to handle some load, especially when several clients are requesting long lists frequently. Push implementations tend to use fewer resources. It is therefore advised to clients *pulling* lists from a server to do this on a relative low polling interval: think in hours, not minutes, and to introduce some splay (randomize the length of the poll interface a bit).

4.1.4. Request format

The request method can be any of GET, POST, PUT, PATCH or DELETE. The OCPI protocol uses them in a way similar to REST APIs.

Method	Description
GET	Fetches objects or information.
POST	Creates new objects or information.
PUT	Updates existing objects or information.
PATCH	Partially updates existing objects or information.
DELETE	Removes existing objects or information.

The HTTP header: Content-Type SHALL be set to `application/json` for any request that contains a message body: POST, PUT and PATCH. When no body is present, probably in a GET or DELETE, then the Content-Type header MAY be omitted.

4.1.4.1. GET

A server is not required to return all objects to a client, the server might for example not send all CDRs to a client, because some CDRs do not belong to this client.

When a client receives objects from the server that contain invalid JSON or invalid OCPI objects (For example: missing fields), the client has no way of letting this know to the server. It is advised to log these errors and contact the server administrator about this. When a list of objects contains some objects that are correct and some with 'problems' the client should at least process the correct OCPI objects.

Pagination

All GET methods that return a list of objects have pagination, this allows a client and server to control the number of objects returned in the response to a GET request, while still enabling the client to retrieve all objects by doing

multiple requests with different parameters. Without pagination, the server has to return all objects in one response that could potentially contain millions of objects.

To enable pagination of the returned list of objects, additional URL parameters are allowed for the GET request and additional headers need to be added to the response.

Paginated Request

The following table lists all the parameters that have to be supported but might be omitted by a client request.

Parameter	Datatype	Description
date_from	DateTime	Only return objects that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	Only return objects that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	The offset of the first object returned. Default is 0 (the first object).
limit	int	The maximum number of objects to GET. The server might decide to return fewer objects, either because there are no more objects, or the server limits the maximum number of objects to return. This is to prevent, for example, overloading the system.

The `date_from` is inclusive and `date_to` exclusive, this way, when sequential requests with to the same end-point are done, the next interval will have no overlap and the `date_from` of the next interval is simply the `date_to` of the previous interval.

Example: With offset=0 and limit=10 the server shall return the first 10 records (if 10 objects match the request). Then the next page starts with offset=10.

Paginated Response

For pagination to work correctly, it is important that multiple calls to the same URL (including query parameters): result in the same objects being returned by the server. For this to be the case, the sequence of objects mustn't change, or as little as possible. It is best practice to return the oldest objects first, that is, order the objects by creation date ascending. While a client crawls over the pages (multiple GET requests every time to the 'next' page Link), a new object might be created on the server. The client detects this: the `X-Total-Count` will be higher on the next call. Even so, the client does not have to retry any requests when this happens because only the last page will be different. This means the client will not be required to crawl all pages all over again. When the client has reached to last page it has retrieved all relevant pages and is up to date.

NOTE

Some query parameters can cause concurrency problems. For example the `date_to` query parameter. When there are for example 1000 objects matching a query for all objects with `date_to` before 2016-01-01. While crawling over the pages one of these objects is updated. The client detects this: `X-Total-Count` will be lower in the next request. It is advised to redo the previous GET with the `offset` lowered by 1 (if the `offset` was not 0) and after that continue crawling the 'next' page links.

HTTP headers that have to be added to any paginated GET response.

HTTP Header	Datatype	Description
Link	String	Link to the 'next' page should be provided when this is NOT the last page. The Link should also contain any filters present in the original request. See the examples below.
X-Total-Count	int	(Custom HTTP Header) The total number of objects available in the server system that match the given query (including the given query parameters, for example: `date_to` and `date_from` but excluding `limit` and `offset`) and that are available to this client. For example: The CPO server might return less CDR objects to an eMSP than the total number of CDRs available in the CPO system.
X-Limit	int	(Custom HTTP Header) The maximum number of objects that the server can return. Note that this is an upper limit. If there are not enough remaining objects to return, fewer objects than this upper limit number will be returned, X-Limit SHALL then still show the upper limit, not the number of objects returned.

NOTE HTTP header names are case-insensitive

Pagination Examples

Example of a required OCPI pagination link header:

```
Link: <https://www.server.com/ocpi/cpo/2.2.1/cdrs/?offset=150&limit=50>; rel="next"
```

After the client has called the given "next" page URL above the Link parameter will most likely look like this:

```
Link: <https://www.server.com/ocpi/cpo/2.2.1/cdrs/?offset=200&limit=50>; rel="next"
```

Example of a query with filters: Client does a GET to:

```
https://www.server.com/ocpi/cpo/2.2.1/cdrs/?date_from=2016-01-01T00:00:00Z&date_to=2016-12-31T23:59:59Z
```

The server should return (when the server has enough objects and the limit is the amount of objects the server wants to send is 100.) *(This example should have been on 1 line, but didn't fit the paper width.)*

```
Link: <https://www.server.com/ocpi/cpo/2.2.1/cdrs/?offset=100
              &limit=100&date_from=2016-01-01T00:00:00Z&date_to=2016-12-31T23:59:59Z>; rel="next"
```

Example of a server limiting the amount of objects returned: Client does a GET to:

```
https://www.server.com/ocpi/cpo/2.2.1/cdrs/?limit=2000
```

The server should return (when the server has enough objects and the limit is the amount of objects the server wants to send is 100.) The `X-Limit` HTTP header should be set to 100 as well.

```
Link: <https://www.server.com/ocpi/cpo/2.2.1/cdrs/?offset=100&limit=100>; rel="next"
```

4.1.4.2. PUT

A PUT request must specify all required fields of an object (similar to a POST request). Optional fields that are not included will revert to their default value which is either specified in the protocol or NULL.

4.1.4.3. PATCH

A PATCH request must only specify the object's identifier (if needed to identify this object) and the fields to be updated. Any fields (both required or optional) that are left out remain unchanged.

The MIME-type of the request body is: `application/json` and may contain the data as documented for each endpoint.

In case a PATCH request fails, the client is expected to call the GET method to check the state of the object in the other party's system. If the object doesn't exist, the client should do a PUT.

4.1.5. Client Owned Object Push

Normal client/server RESTful services work in a way where the Server is the owner of the objects that are created. The client requests a POST method with an object to the end-point URL. The response sent by the server will contain the URL to the new object. The client will request only one server to create a new object, not multiple servers.

Many OCPI modules work differently: the client is the owner of the object and only pushes the information to one or more servers for information sharing purposes. For example the CPO owns the Tariff objects and pushes them to a couple of eMSPs, so each eMSP gains knowledge of the tariffs that the CPO will charge them for their customers' sessions. eMSP might receive Tariff objects from multiple CPOs. They need to be able to make a distinction between the different tariffs from different CPOs.

The distinction between objects from different CPOs/eMSPs is made based on a {country_code} and {party_id}. The country_code's and party_id's of the parties on the other platform are received during the credentials handshake in the CredentialsRoles. The roles exchanged during the credentials handshake provide the server with details needed to determine which URLs a client might use.

Client Owned Object URL definition: {base-ocpi-url}/{end-point}/{country-code}/{party-id}/{object-id}

Example of a URL to a Client Owned Object

```
https://www.server.com/ocpi/cpo/2.2.1/tariffs/NL/TNM/14
```

POST is not supported for these kinds of modules. PUT is used to send new objects to the servers.

To identify the owner of data, the party generating the information that is provided to other parties via OCPI, a 'Data owner' is provided at the beginning of every module that has a clear owner.

4.1.5.1. Errors

When a client tries to access an object with a URL that has a different country_code and/or party_id than one of the CredentialsRoles given during the credentials handshake, it is allowed to respond with an HTTP `404` status code, this way blocking client access to objects that do not belong to them.

When a client pushes a Client Owned Object, but the {object-id} in the URL is different from the id in the object being pushed, server implementations are advised to return an OCPI status code: 2001.

4.1.6. Client Owned Object Pull

When doing a GET on the Sender interface of a module, the owner of an object can be determined by looking at the {country_code} and {party_id} in the object itself.

When one or more objects, returned in the response, do not meet one of the CredentialsRoles given during the credentials handshake, these objects may be ignored.

4.1.7. Response format

The content that is sent with all the response messages is an 'application/json' type and contains a JSON object with the following properties:

Property	Type	Card.	Description
data	Array or Object or String	* or ?	Contains the actual response data object or list of objects from each request, depending on the cardinality of the response data, this is an array (card. * or +), or a single object (card. 1 or ?)
status_code	int	1	OCPI status code, as listed in Status Codes, indicates how the request was handled. To avoid confusion with HTTP codes, OCPI status codes consist of four digits.
status_message	string	?	An optional status message which may help when debugging.
timestamp	DateTime	1	The time this message was generated.

For brevity's sake, any further examples used in this specification will only contain the value of the "data" field. In reality, it will always have to be wrapped in the above response format.

When a request cannot be accepted, the type response depends on the type of error. For more information see: Status codes

For errors on the HTTP layer, use HTTP error response codes, including the response format above, that contains more details. HTTP status codes are described on w3.org.

NOTE

Earlier versions of the OCPI 2.2.1 did not clearly specify what should be in the `data` field of the response format for every request. We advise that in cases where the specification does not explicitly specify what to put in the `data` field for the response to a certain request, the platform receiving the response accept both the `data` field being absent and the data field being present with any possible value. We also advise that in such cases, platform sending the response leave the `data` field unset in the response format. This applies for example to PUT requests when pushing Session objects, and PATCH requests to add charging periods to Sessions.

4.1.7.1. Example: Version information response (list of objects)

```
{
  "data": [{
```

```
    "version": "2.1.1",
    "url": "https://example.com/ocpi/cpo/2.1.1"
  }, {
    "version": "2.2",
    "url": "https://example.com/ocpi/cpo/2.2"
  }],
  "status_code": 1000,
  "status_message": "Success",
  "timestamp": "2015-06-30T21:59:59Z"
}
```

4.1.7.2. Example: Version details response (one object)

```
{
  "data": {
    "version": "2.2",
    "endpoints": [{
      "identifier": "credentials",
      "role": "SENDER",
      "url": "https://example.com/ocpi/cpo/2.2/credentials"
    }, {
      "identifier": "locations",
      "role": "SENDER",
      "url": "https://example.com/ocpi/cpo/2.2/locations"
    }]
  },
  "status_code": 1000,
  "status_message": "Success",
  "timestamp": "2015-06-30T21:59:59Z"
}
```

4.1.7.3. Example: Tokens GET Response with one Token object. (CPO end-point) (one object)

```
{
  "data": {
    "country_code": "DE",
    "party_id": "TNM",
    "uid": "012345678",
    "type": "RFID",
    "contract_id": "FA54320",
    "visual_number": "DF000-2001-8999",
    "issuer": "TheNewMotion",
    "valid": true,
    "whitelist": "ALLOWED",
    "last_updated": "2015-06-29T22:39:09Z"
  },
  "status_code": 1000,
  "status_message": "Success",
  "timestamp": "2015-06-30T21:59:59Z"
}
```

4.1.7.4. Example: Tokens GET Response with list of Token objects. (eMSP end-point) (list of objects)

```
{
  "data": [{
    "country_code": "NL",
    "party_id": "TNM",
    "uid": "100012",
    "type": "RFID",
    "contract_id": "FA54320",
```

```
    "visual_number": "DF000-2001-8999",
    "issuer": "TheNewMotion",
    "valid": true,
    "whitelist": "ALWAYS",
    "last_updated": "2015-06-21T22:39:05Z"
  }, {
    "country_code": "NL",
    "party_id": "TNM",
    "uid": "100013",
    "type": "RFID",
    "contract_id": "FA543A5",
    "visual_number": "DF000-2001-9000",
    "issuer": "TheNewMotion",
    "valid": true,
    "whitelist": "ALLOWED",
    "last_updated": "2015-06-28T11:21:09Z"
  }, {
    "country_code": "NL",
    "party_id": "TNM",
    "uid": "100014",
    "type": "RFID",
    "contract_id": "FA543BB",
    "visual_number": "DF000-2001-9010",
    "issuer": "TheNewMotion",
    "valid": false,
    "whitelist": "ALLOWED",
    "last_updated": "2015-05-29T10:12:26Z"
  }],
  "status_code": 1000,
  "status_message": "Success",
  "timestamp": "2015-06-30T21:59:59Z"
}
```


4.1.7.5. Example: Response with an error (contains no data field)


```
{
    "status_code": 2001,
    "status_message": "Missing required field: type",
    "timestamp": "2015-06-30T21:59:59Z"
}
```


4.1.8. Non-specified JSON fields

An OCPI Platform SHALL NOT reject request or response payloads based on the presence of JSON object field names that are not documented in this specification.

OCPI implementers are encouraged to extend OCPI with new fields to address needs that are not foreseen by the specification and to adhere to the spirit of RFC 6648 when doing so.

When extending OCPI with such non-specified fields, implementers:

- SHOULD assume that their extensions will eventually be incorporated into OCPI or otherwise become widely used,
- SHOULD choose meaningful field names that are currently unused to the best of their knowledge,
- SHOULD NOT use name prefixes like "x-" or "custom" to indicate the non-specified nature of these field names, and
- SHOULD consult further guidance on extending OCPI at https://evroaming.org/extending-ocpi/.

4.1.9. Message Routing

When the development of OCPI was started, it was designed for peer-to-peer communication between CPO and eMSP. This has advantages, but also disadvantages. Having to set up and maintain OCPI connections to a lot of parties requires more effort than doing it for only a couple of connections. By communication via one or more Hubs, the amount of OCPI connections is reduced, while still being able to offer roaming to a lot of different parties and customers.

With the introduction of Message Routing, OCPI is now better usable for communication via Hubs.

All examples/sequence diagrams in this section use the roles CPO and eMSP as examples, they could be switched, it could be other roles.

4.1.9.1. Platforms

With Message Routing functionality it also becomes possible to support Platforms that host multiple roles. A lot of parties are not only CPO or eMSP. Most are both CPO and eMSP. Some parties are doing business in multiple countries, which means to operate with different `country_codes` Some parties have a platform on which the host service for other CPOs/eMSPs. Some parties are themselves CPO and host CPO services for others, but other parties are (themselves) not a CPO or other role in the EV charging landscape but do provide service to CPOs/eMSPs, etc.

4.1.9.2. Message Routing Headers

When OCPI is used to communicate to/from a Platform or via a Hub (which is the most common usage of OCPI, only exception is a peer-to-peer connection between two parties that both have only one OCPI party and role implemented.) the following four HTTP headers are to be added to any request/response to allow messages to be routed.

When implementing OCPI these four headers SHALL be implemented for any request/response to/from a Functional Module. This does not mean they have to be present in all request. There are situation/special request where some headers can or shall be omitted, See: Open Routing Request

Only requests/responses from Function Modules: such as: Tokens, Locations, CDRs etc. SHALL be routed, so need the routing headers.

The requests/responses to/from Configuration Modules: Credentials, Versions and Hub Client Info are not to be routed, and are for Platform-to-Platform or Platform-to-Hub communication. Thus routing headers SHALL NOT be used with these modules.

HTTP Header	Datatype	Description
OCPI-to-party-id	CiString(3)	'party id' of the connected party this message is to be sent to.
OCPI-to-country-code	CiString(2)	'country code' of the connected party this message is to be sent to.
OCPI-from-party-id	CiString(3)	'party id' of the connected party this message is sent from.
OCPI-from-country-code	CiString(2)	'country code' of the connected party this message is sent from.

NOTE HTTP header names are case-insensitive

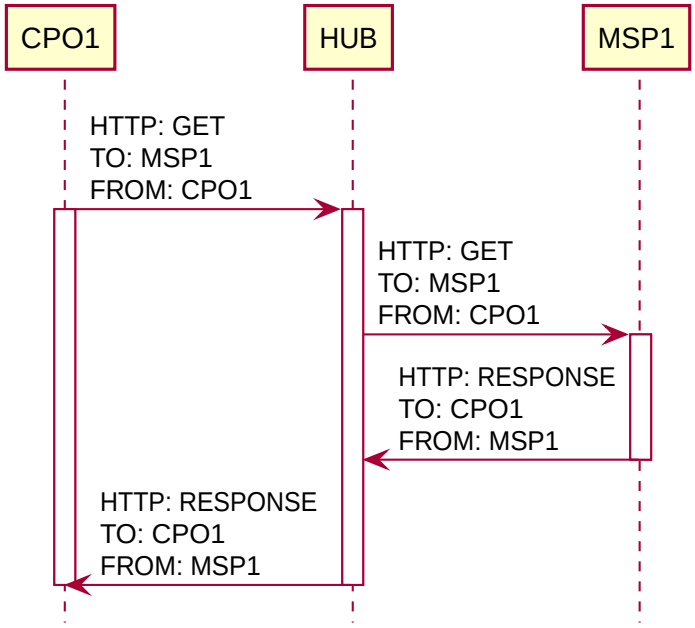


Figure 10. Example sequence diagram of a GET for 1 Object from a CPO to an eMSP.

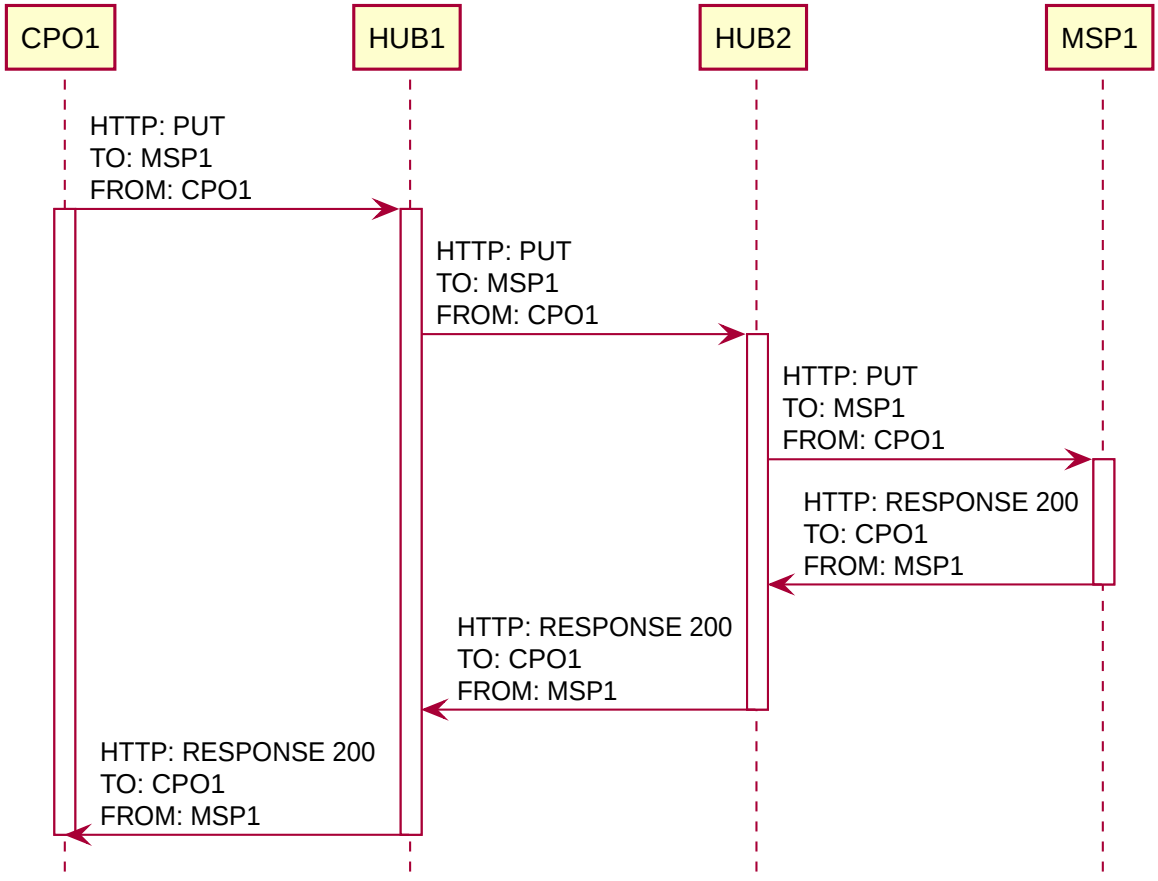


Figure 11. Example sequence diagram of a PUT via 2 Hubs.

4.1.9.3. Broadcast Push

For simplicity, connected clients might push (POST, PUT, PATCH) information to all connected clients with an "opposite role", for example: CPO pushing information to all eMSPs and NSPs, eMSP pushing information to all CPOs. (The role "Other" is seen as an eMSP type of role, so Broadcast Push from a CPO is also sent to "Other". Messages

from "Other" are only sent to CPOs and not to eMSPs though.)

When using Broadcast Push, the Hub broadcasts received information to all connected clients. To send data through a Hub might be very useful to share information like Locations or Tokens with all parties connected to the Hub that have implemented the corresponding module. This means only one request to the Hub will be necessary, as all connected clients will be served by the Hub.

To send a Broadcast Push, the client uses the party-id and country-code of the Hub in the 'OCPI-to-' headers. The Hub parses the request and sends a response to the client, which optionally contains its own party-id and country-code in the 'OCPI-from-' headers. The Hub then sends the pushed data to any client implementing the corresponding applicable module, using its own party-id and country-code in the 'OCPI-from-' headers. The client receiving a Push from a Hub (with the Hubs information in the 'OCPI-from-' headers) will respond to this Push with the Hubs party-id and country-code in the 'OCPI-to-' headers.

GET SHALL NOT be used in combination with Broadcast Push. If the requesting party wants to GET information of which it does not know the receiving party, an Open Routing Request MUST be used. (see below)

Broadcast Push SHALL only be used with information that is meant to be sent to all other parties. It is useful to share data like Tokens and Locations, but not so much for CDRs and Sessions as these pieces of information are specific to only one party and are possibly even protected by GDPR or other laws.

NOTE	For "Client Owned Objects", the party-id and country-code in the URL segments will still be the original party-id and country-code from the original client sending the Broadcast Push to the Hub.

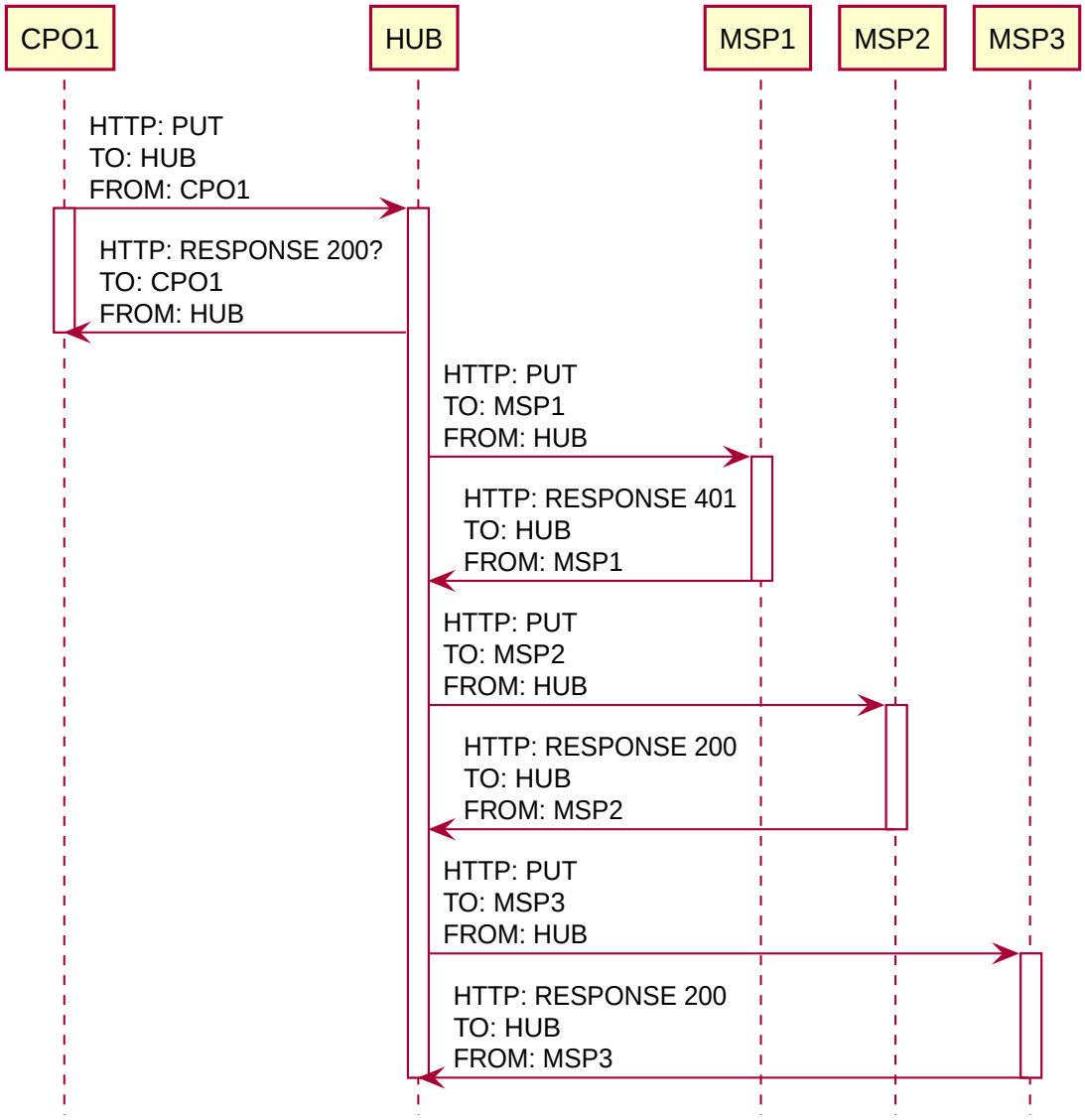


Figure 12. Example sequence diagram of a Broadcast Push from a CPO to multiple eMSPs.

4.1.9.4. Open Routing Request

When a Hub has the intelligence to route messages based on the content of the request, or the requesting party does not know the destination of a request, the 'OCPI-to-' headers can be omitted in the request towards the Hub. The Hub can then decide to which party a request needs to be routed, or that it needs to be broadcasted if the destination cannot be determined.

This has nothing to do with Broadcast Push though, as Broadcast Push only works for the Push model, not for GET requests.

Open Routing Requests are possible for GET (Not GET ALL), POST, PUT, PATCH and DELETE.

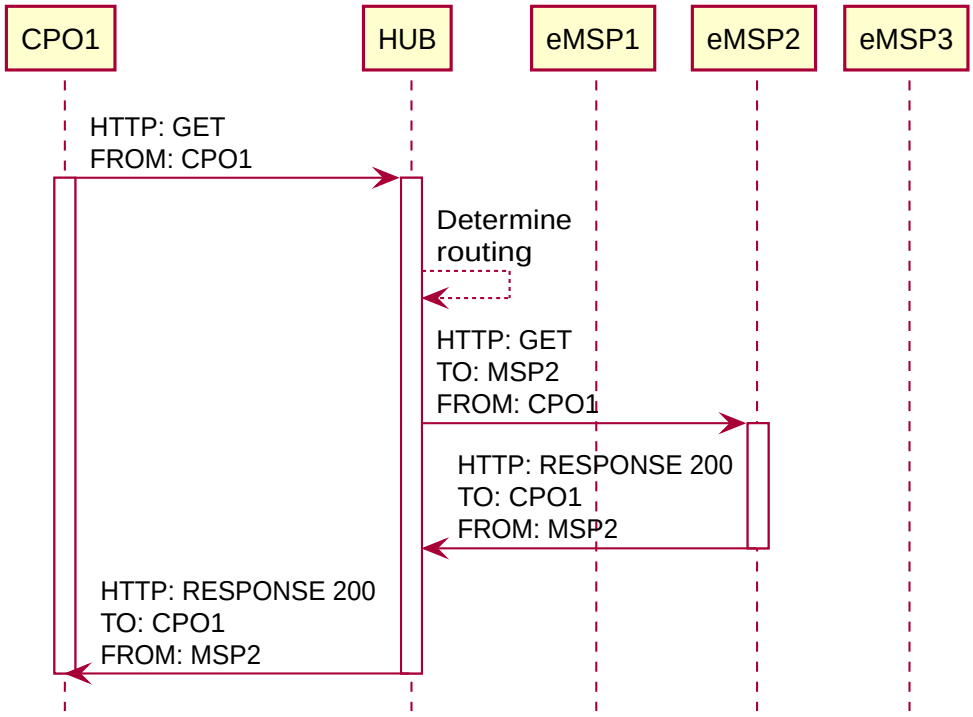


Figure 13. Example sequence diagram of a open routing GET from a CPO via the Hub.

4.1.9.5. GET All via Hubs

A client (Receiver) can request a GET on the Sender interface of a module implemented by a Hub. To request a GET All from a Hub, the client uses the party-id and country-code of the Hub in the 'OCPI-to-' headers, and calls the GET method on the Sender interface of a module.

The Hub can then combine objects from different connected parties and return them to the client.

The client can determine the owner of the objects by looking at the `county_code` and `party_id` in the individual objects returned by the hub.

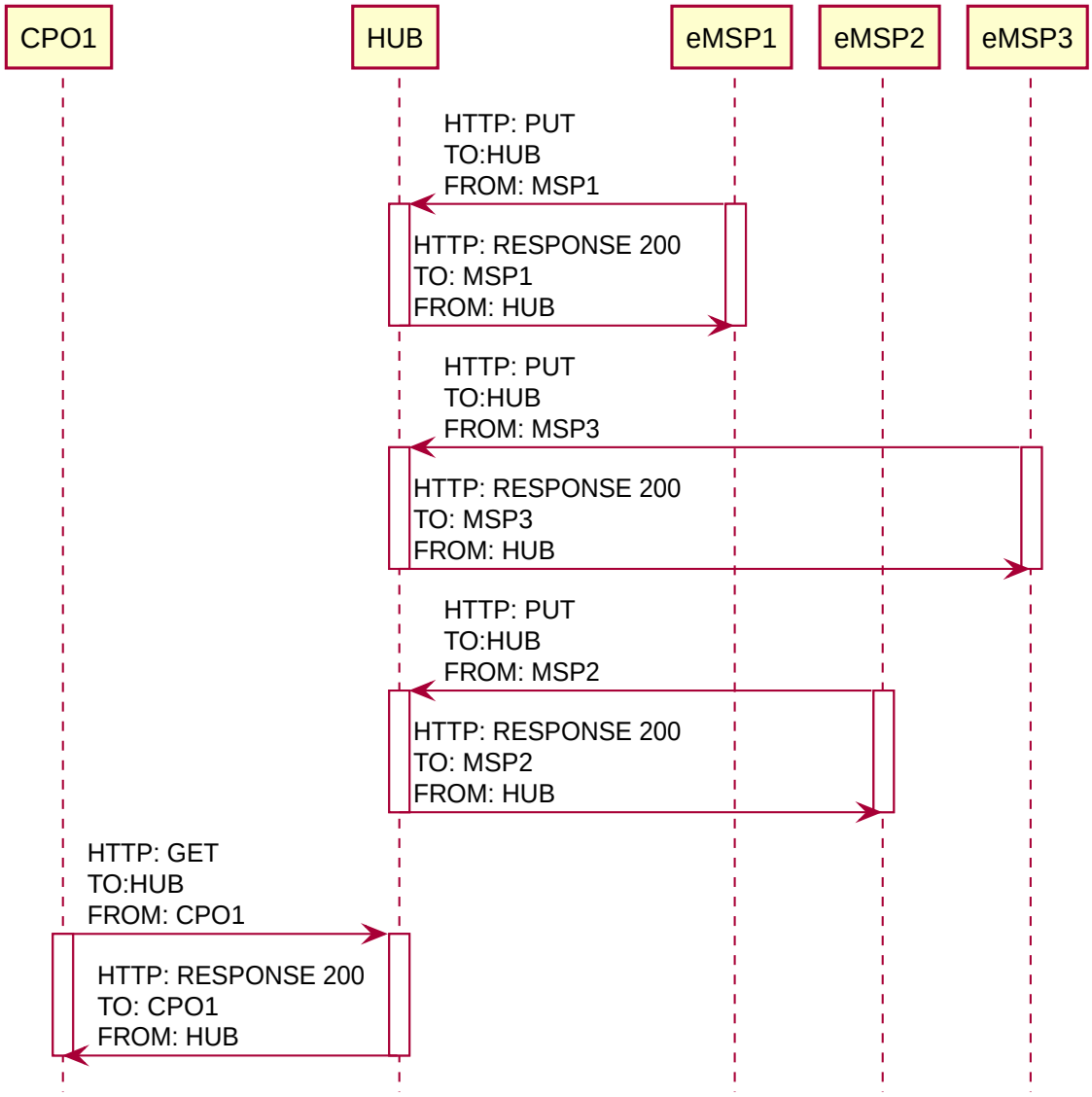


Figure 14. Example sequence diagram of a GET All via the Hub, .

4.1.9.6. Overview of required/optional routing headers for different scenarios

The following section shows which headers are required/optional and which 'OCPI-to-'/'OCPI-from-' IDs need to be used.

This is not an exclusive list, combinations are possible.

Party to Party (without Hub)

This table contains the description of which headers are required to be used for which message when a request is sent directly from one platform provider to another platform provider, without a Hub in between. The headers are addressing the parties to/from which the message is sent, not the platform itself.

Name	Route	TO Headers	FROM Headers
Direct request	Requesting platform provider to Receiving platform provider	Receiving-party	Requesting-party
Direct response	Receiving platform provider to Requesting platform provider	Requesting-party	Receiving-party

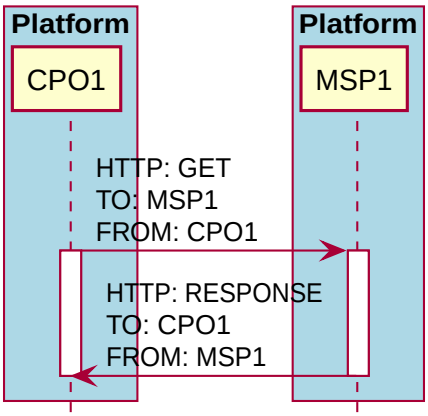


Figure 15. Example sequence diagram of a GET for 1 Object from a CPO on one platform to an MSP on another platform directly (without a Hub)

Party to Party via Hub

This table contains the description of which headers are required to be used for which message when a request is routed from one platform to another platform via a Hub.

Name	Route	TO Headers	FROM Headers
Direct request	Requesting platform to Hub	Receiving-party	Requesting-party
Direct request	Hub to receiving platform	Receiving-party	Requesting-party
Direct response	Receiving platform to Hub	Requesting-party	Receiving-party
Direct response	Hub to requesting platform	Requesting-party	Receiving-party

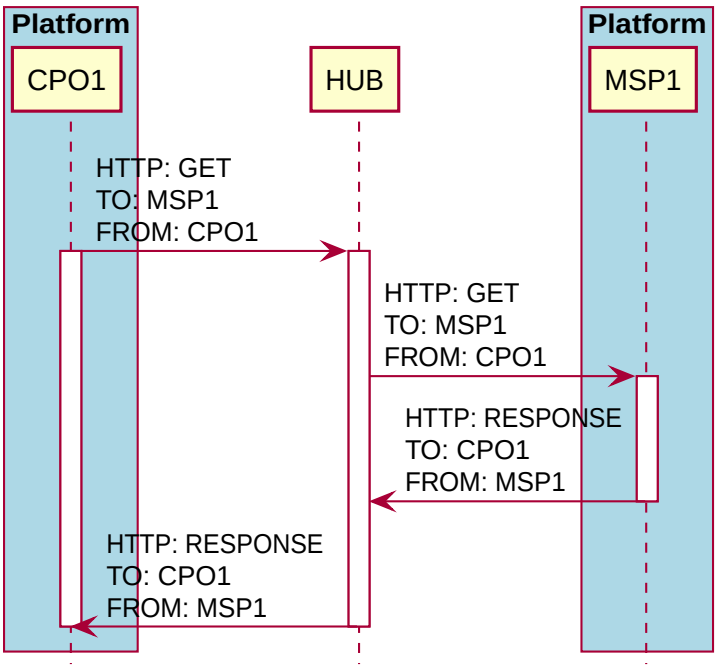


Figure 16. Example sequence diagram of a GET for 1 Object from one Platform to another Platform via a Hub

Party to Party Broadcast Push

This table contains the description of which headers are required to be used for which message when a request is a Broadcast Push to the Hub.

Name	Route	TO Headers	FROM Headers
Broadcast request	Requesting platform to Hub	Hub	Requesting-party
Broadcast response	Hub to requesting platform	Requesting-party	Hub
Broadcast request	Hub to receiving platform	Receiving-party	Hub
Broadcast response	Receiving platform to Hub	Hub	Receiving-party

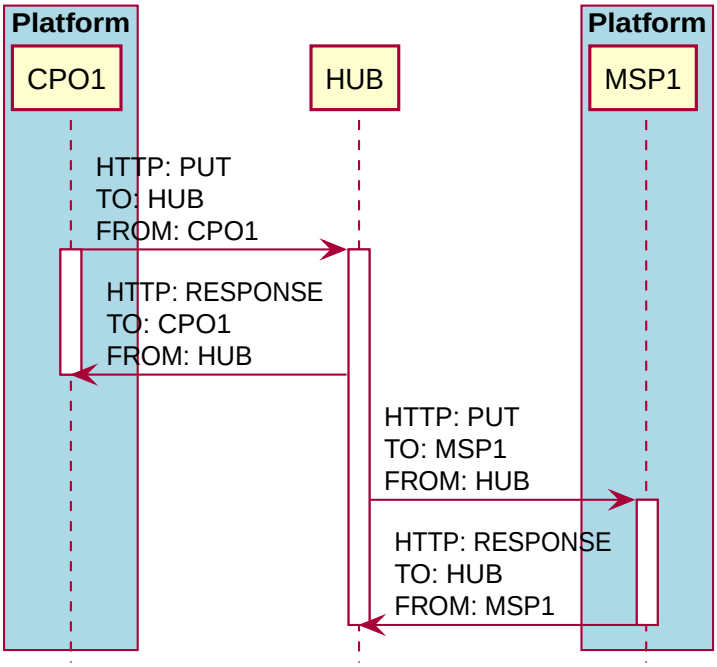


Figure 17. Example sequence diagram of Broadcast Push from one Platform to another Platform via a Hub

Party to Party Open Routing Request

This table contains the description of which headers are required to be used for which message when the routing of a request needs to be determined by the Hub itself. For an Open Routing Request, the TO headers in the request from the requesting party to the Hub MUST be omitted.

Name	Route	TO Headers	FROM Headers
Open request	Requesting platform to Hub		Requesting-party
Open request	Hub to receiving platform	Receiving-party	Requesting-party
Open response	Receiving platform to Hub	Requesting-party	Receiving-party
Open response	Hub to requesting platform	Requesting-party	Receiving-party

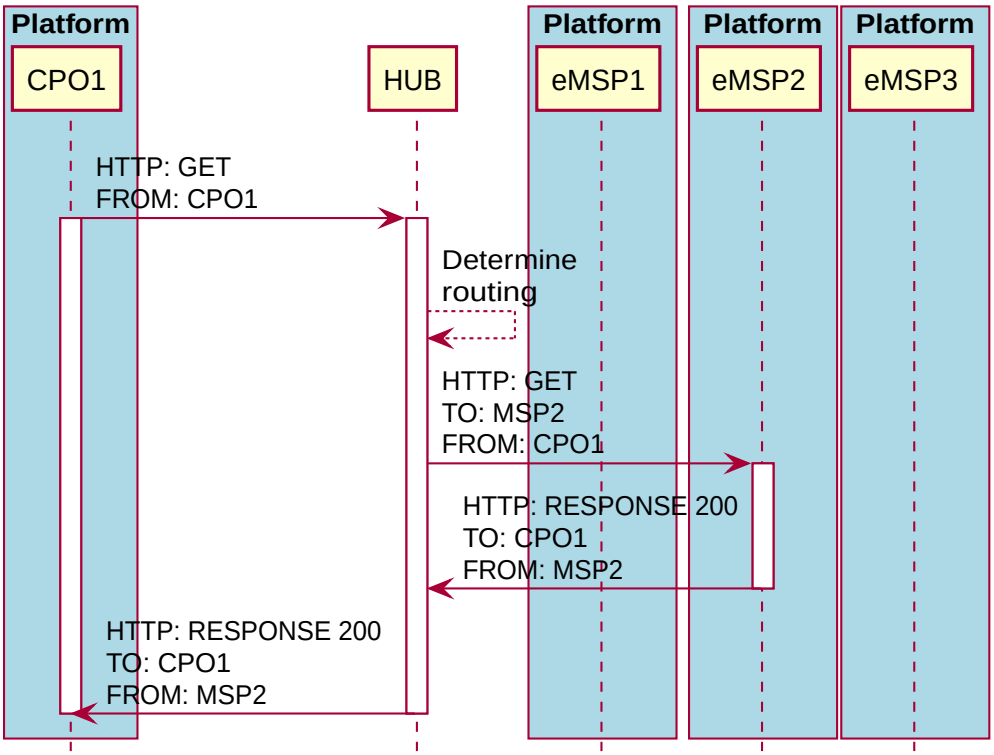


Figure 18. Example sequence diagram of a open routing between platforms GET from a CPO via the Hub

4.1.9.7. GET All via Hubs

This table contains the description of which headers are required to be used when doing a GET All via a Hub. For a GET All via Hub: The HTTP Method SHALL be GET, The call is to a Senders Interface, the TO headers in the request to the Hub has to be set to the Hub.

Name	Route	TO Headers	FROM Headers
GET All via Hubs request	Requesting platform to Hub	Hub	Requesting-party
GET All via Hubs response	Hub to receiving platform	Requesting-party	Hub

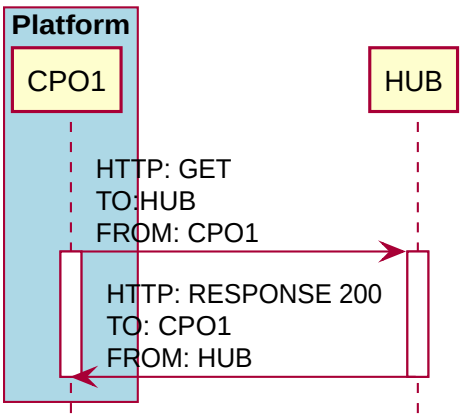


Figure 19. Example sequence diagram of a GET All via the Hub, .

4.1.9.8. Timestamps and Objects send via Hubs

When OCPI Objects are sent via Hubs, the `last_updated` fields SHALL NOT be updated by the Hub.

4.1.10. No data available

There are rare situation, probably use cases not foreseen by the team developing OCPI, where a certain field, that is required, cannot be filled. In such cases, and only in such cases, it is allowed to set a string field to the value: `#NA`.

`#NA` is not allowed to be used when a party does not have or want to provide the data, but is able to provide the data when they would spend time/resources to get/provide the data.

4.2. Unique message IDs

For debugging issues, OCPI implementations are required to include unique IDs via HTTP headers in every request/response.

HTTP Header	Description
X-Request-ID	Every request SHALL contain a unique request ID, the response to this request SHALL contain the same ID.
X-Correlation-ID	Every request/response SHALL contain a unique correlation ID, every response to this request SHALL contain the same ID.

NOTE HTTP header names are case-insensitive

It is advised to used GUID/UUID as values for X-Request-ID and X-Correlation-ID.

When a Hub forwards a request to a party, the request to this party SHALL contain a new unique value in the X-Request-ID HTTP header, not a copy of the X-Request-ID HTTP header taken from the incoming request that is being forwarded.

When a Hub forwards a request to a party, the request SHALL contain the same X-Correlation-ID HTTP header (with the same value).

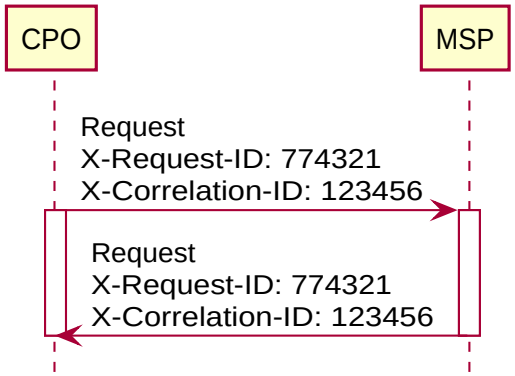


Figure 20. Example sequence diagram of the uses of X-Request-ID and X-Correlation-ID in a peer-to-peer topology.

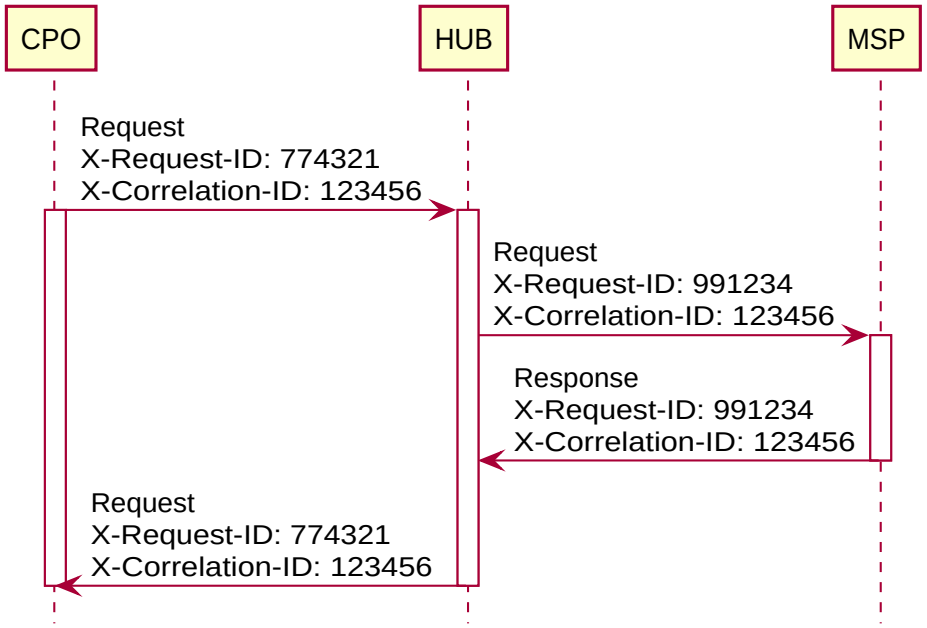


Figure 21. Example sequence diagram of the uses of X-Request-ID and X-Correlation-ID in a topology with a Hub.

4.3. Interface endpoints

As OCPI contains multiple interfaces. Different endpoints are available for messaging. The protocol is designed such that the exact URLs of the endpoints can be defined by each party. It also supports an interface per version.

The locations of all the version-specific endpoints can be retrieved by fetching the API information from the versions endpoint. Each version-specific endpoint will then list the available endpoints for that version. It is strongly recommended to insert the protocol version into the URL.

For example: `/ocpi/cpo/2.2.1/locations` and `/ocpi/emsp/2.2.1/locations`.

The URLs of the endpoints in this document are descriptive only. The exact URL can be found by fetching the endpoint information from the API info endpoint and looking up the identifier of the endpoint.

Operator interface	**Identifier**	**Example URL**
Credentials	credentials	`https://example.com/ocpi/cpo/2.2.1/credentials`
Charging location details	locations	`https://example.com/ocpi/cpo/2.2.1/locations`

eMSP interface	**Identifier**	**Example URL**
Credentials	credentials	`https://example.com/ocpi/emsp/2.2.1/credentials`
Charging location updates	locations	`https://example.com/ocpi/emsp/2.2.1/locations`

4.4. Offline behaviour

During communication over OCPI, one of the communicating parties might be unreachable for an undefined amount of time. OCPI works event-based, new messages and status are pushed from one party to another. When communication is lost, updates cannot be delivered.

OCPI messages SHOULD NOT be queued. When a client does a POST, PUT or PATCH request and that request fails or times out, the client should not queue the message and retry the same message again later.

When the connection is re-established, it is up to the target-server of a connection to GET the current status from to source-server to get back to a synchronized state.

For example:

- CDRs of the period of communication loss can be retrieved with a GET command on the CDRs module, with filters to retrieve only CDRs of the period since the last CDR has been received.
- Status of EVSEs (or Locations) can be retrieved by calling a GET on the Locations module.

5. Status codes

There are two types of status codes:

- Transport related (HTTP)
- Content related (OCPI)

The transport layer ends after a message is correctly parsed into a (semantically unvalidated) JSON structure. When a message does not contain a valid JSON string, the HTTP error `400 - Bad request` MUST be returned.

If a request is syntactically valid JSON and addresses an existing resource, a HTTP error MUST NOT be returned. Those requests are supposed to have reached the OCPI layer.

In case of a GET request, when the resource does NOT exist, the server SHOULD return a HTTP `404 - Not Found`.

When the server receives a valid OCPI object it SHOULD respond with:

- HTTP `200 - Ok` when the object already existed and has successfully been updated.
- HTTP `201 - Created` when the object has been newly created in the server system.

Requests that reach the OCPI layer SHOULD return an OCPI response message with a `status_code` field as defined below.

Custom status code range values SHALL NOT be used by standard OCPI module as described in this document! When custom status codes are used, keep in mind that different custom modules could use the same values with a different meaning, as they are not standardized.

Range	Description
1xxx	Success
2xxx	Client errors – The data sent by the client can not be processed by the server
3xxx	Server errors – The server encountered an internal error

When the status code is in the success range (1xxx), the `data` field in the response message SHOULD contain the information as specified in the protocol. Otherwise the `data` field is unspecified and MAY be omitted, set to `null` or something else that could help to debug the problem from a programmer's perspective. For example, it could specify which fields contain an error or are missing.

5.1. 1xxx: Success

Code	Description
1000	Generic success code
19xx	Reserved range for custom success status codes (1900-1999).

5.2. 2xxx: Client errors

Errors detected by the server in the message sent by a client where the client did something wrong.

Code	Description
2000	Generic client error
2001	Invalid or missing parameters , for example: missing `last_updated` field in a PATCH request.
2002	Not enough information, for example: Authorization request with too little information.
2003	Unknown Location, for example: Command: START_SESSION with unknown location.
2004	Unknown Token, for example: 'real-time' authorization of an unknown Token.
29xx	Reserved range for custom client error status codes (2900-2999).

5.3. 3xxx: Server errors

Error during processing of the OCPI payload in the server. The message was syntactically correct but could not be processed by the server.

Code	Description
3000	Generic server error
3001	Unable to use the client's API. For example during the credentials registration: When the initializing party requests data from the other party during the open POST call to its credentials endpoint. If one of the GETs can not be processed, the party should return this error in the POST response.
3002	Unsupported version
3003	No matching endpoints or expected endpoints missing between parties. Used during the registration process if the two parties do not have any mutual modules or endpoints available, or the minimal implementation expected by the other party is not been met.
39xx	Reserved range for custom server error status codes (3900-3999).

5.4. 4xxx: Hub errors

When a server encounters an error, client side error (2xxx) or server side error (3xxx), it sends the status code to the Hub. The Hub SHALL then forward this error to the client which sent the request (when the request was not a Broadcast Push).

For errors that a Hub encounters while routing messages, the following OCPI status codes shall be used.

Code	Description
4000	Generic error
4001	Unknown receiver (TO address is unknown)
4002	Timeout on forwarded request (message is forwarded, but request times out)
4003	Connection problem (receiving party is not connected)
49xx	Reserved range for custom hub error status codes (4900-4999).

6. *Versions* module

Type: Configuration Module

This is the required base module of OCPI. This module is the starting point for any OCPI connection. Via this module, clients can learn which versions of OCPI a server supports, and which modules it supports for each of the versions.

6.1. Version information endpoint

This endpoint lists all the available OCPI versions and the corresponding URLs to where version specific details such as the supported endpoints can be found.

Endpoint structure definition:

No structure defined. This is open for every party to define themselves.

Examples:

`https://www.server.com/ocpi/cpo/versions`

`https://www.server.com/ocpi/emsp/versions`

`https://ocpi.server.com/versions`

The exact URL to the implemented version endpoint should be given (offline) to parties that want to communicate with your OCPI implementation.

Both, CPOs and eMSPs MUST implement such a version endpoint.

Method	Description
GET	Fetch information about the supported versions.

6.1.1. Data

Type	Card.	Description
Version	+	A list of supported OCPI versions.

6.1.2. Version *class*

Property	Type	Card.	Description
version	VersionNumber	1	The version number.
url	URL	1	URL to the endpoint containing version specific information.

6.1.3. GET

Fetch all supported OCPI versions of this CPO or eMSP.

6.1.3.1. Example

```
[
  {
    "version": "2.1.1",
    "url": "https://www.server.com/ocpi/2.1.1"
  },
  {
    "version": "2.2.1",
    "url": "https://www.server.com/ocpi/2.2.1"
  }
]
```

6.2. Version details endpoint

Via the version details, the parties can exchange which modules are implemented for a specific version of OCPI, which interface role is implemented, and what the endpoint URL is for this interface.

Parties that are both CPO and eMSP (or a Hub) can implement one version endpoint that covers both roles. With the information that is available in the version details, parties don't need to implement a separate endpoint per role (CPO or eMSP) anymore. In practice this means that when a company is both a CPO and an eMSP and it connects to another party that implements both interfaces, only one OCPI connection is needed.

NOTE	OCPI 2.2 introduced the role field in the version details. Older versions of OCPI do not support this.

Endpoint structure definition:

No structure defined. This is open for every party to define themselves.

Examples:

`https://www.server.com/ocpi/cpo/2.2.1`

`https://www.server.com/ocpi/emsp/2.2.1`

`https://ocpi.server.com/2.2.1/details`

This endpoint lists the supported endpoints and their URLs for a specific OCPI version. To notify the other party that the list of endpoints of your current version has changed, you can send a PUT request to the corresponding credentials endpoint (see the credentials chapter).

Both the CPO and the eMSP MUST implement this endpoint.

Method	**Description**
GET	Fetch information about the supported endpoints for this version.

6.2.1. Data

Property	**Type**	**Card.**	**Description**
version	VersionNumber	1	The version number.

Property	Type	Card.	Description
endpoints	Endpoint	+	A list of supported endpoints for this version.

6.2.2. Endpoint *class*

Property	Type	Card.	Description
identifier	ModuleID	1	Endpoint identifier.
role	InterfaceRole	1	Interface role this endpoint implements.
url	URL	1	URL to the endpoint.

NOTE for the `credentials` module, the value of the role property is not relevant as this module is the same for all roles. It is advised to send "SENDER" as the InterfaceRole for one's own credentials endpoint and to disregard the value of the role property of the Endpoint object for other platforms' credentials modules.

6.2.3. InterfaceRole *enum*

Value	Description
SENDER	Sender Interface implementation. Interface implemented by the owner of data, so the Receiver can Pull information from the data Sender/owner.
RECEIVER	Receiver Interface implementation. Interface implemented by the receiver of data, so the Sender/owner can Push information to the Receiver.

6.2.4. ModuleID *OpenEnum*

The Module identifiers for each endpoint are described in the beginning of each *Module* chapter. The following table contains the list of modules in this version of OCPI. Most modules (except Credentials & Registration) are optional, but there might be dependencies between modules. If there are dependencies between modules, it will be mentioned in the affected module description.

Module	ModuleID	Remark
CDRs	cdrs	
Charging Profiles	chargingprofiles	
Commands	commands	
Credentials & Registration	credentials	Required for all implementations. The `role` field has no function for this module.
Hub Client Info	hubclientinfo	
Locations	locations	

Module	ModuleID	Remark
Sessions	sessions	
Tariffs	tariffs	
Tokens	tokens	

6.2.5. VersionNumber *OpenEnum*

List of known versions.

Value	Description
2.0	OCPI version 2.0
2.1	OCPI version 2.1 (DEPRECATED, do not use, use 2.1.1 instead)
2.1.1	OCPI version 2.1.1
2.2	OCPI version 2.2 (DEPRECATED, do not use, use 2.2.1 instead)
2.2.1	OCPI version 2.2.1
2.3.0	OCPI version 2.3.0 (this version)

6.2.5.1. Custom Modules

Parties are allowed to create custom modules or customized versions of the existing modules. To do so, the ModuleID enum can be extended with additional custom moduleIDs. These custom moduleIDs MAY only be sent to parties with which there is an agreement to use a custom module. Do NOT send custom moduleIDs to parties you are not 100% sure will understand the custom moduleIDs. It is advised to use a prefix (e.g. country-code + party-id) for any custom moduleID, this ensures that the moduleID will not be used for any future module of OCPI.

For example: `nltnm-tokens`

6.2.6. GET

Fetch information about the supported endpoints and their URLs for this OCPI version.

6.2.6.1. Examples

Simple version details example: CPO with only 2 modules.

```
{
  "version": "2.2",
  "endpoints": [
    {
      "identifier": "credentials",
      "role": "SENDER",
      "url": "https://example.com/ocpi/2.2/credentials"
    },
    {
      "identifier": "locations",
      "role": "SENDER",
      "url": "https://example.com/ocpi/cpo/2.2/locations"
    }
  ]
```

```
}
```

Simple version details example: party with both CPO and eMSP with only 2 modules.

In this case the `credentials` module is not defined twice as this module is the same for all roles.

```
{
  "version": "2.2",
  "endpoints": [
    {
      "identifier": "credentials",
      "role": "RECEIVER",
      "url": "https://example.com/ocpi/2.2/credentials"
    },
    {
      "identifier": "locations",
      "role": "SENDER",
      "url": "https://example.com/ocpi/cpo/2.2/locations"
    },
    {
      "identifier": "tokens",
      "role": "RECEIVER",
      "url": "https://example.com/ocpi/cpo/2.2/tokens"
    },
    {
      "identifier": "locations",
      "role": "RECEIVER",
      "url": "https://example.com/ocpi/msp/2.2/locations"
    },
    {
      "identifier": "tokens",
      "role": "SENDER",
      "url": "https://example.com/ocpi/msp/2.2/tokens"
    }
  ]
}
```

7. *Credentials* module

Module Identifier: `credentials`

Type: Configuration Module

The credentials module is used to exchange the credentials token that has to be used by parties for authorization of requests.

Every OCPI request is required to contain a credentials token in the HTTP Authorization header.

7.1. Use cases

7.1.1. Registration

To start using OCPI, the Platforms will need to exchange credentials tokens.

To start the exchange of credentials tokens, one platform has to be selected as Sender for the Credentials module. This has to be decided between the Platforms (outside of OCPI) before they first connect.

To start the credentials exchange, the Receiver Platform must create a unique credentials token: `CREDENTIALS_TOKEN_A` that has to be used to authorize the Sender until the credentials exchange is finished. This credentials token along with the versions endpoint SHOULD be sent to the Sender in a secure way that is outside the scope of this protocol.

The Sender starts the registration process, retrieves the version information and details (using `CREDENTIALS_TOKEN_A` in the HTTP Authorization header). The Sender generates a unique credentials token: `CREDENTIALS_TOKEN_B`, sends it to the Receiver in a POST request to the `credentials` module of the Receiver. The Receiver stores `CREDENTIALS_TOKEN_B` and uses it for any requests to the Sender Platform, including the version information and details.

The Receiver generates a unique credentials token: `CREDENTIALS_TOKEN_C` and returns it to the Sender in the response to the POST request from the Sender.

After the credentials exchange has finished, the Sender SHALL use `CREDENTIALS_TOKEN_C` in future OCPI request to the Receiver Platform. The `CREDENTIALS_TOKEN_A` can then be thrown away, it MAY no longer be used.

(In the sequence diagrams below we use relative paths as short resource identifiers to illustrate API endpoints; please note that they should be absolute URLs in any working implementation of OCPI.)

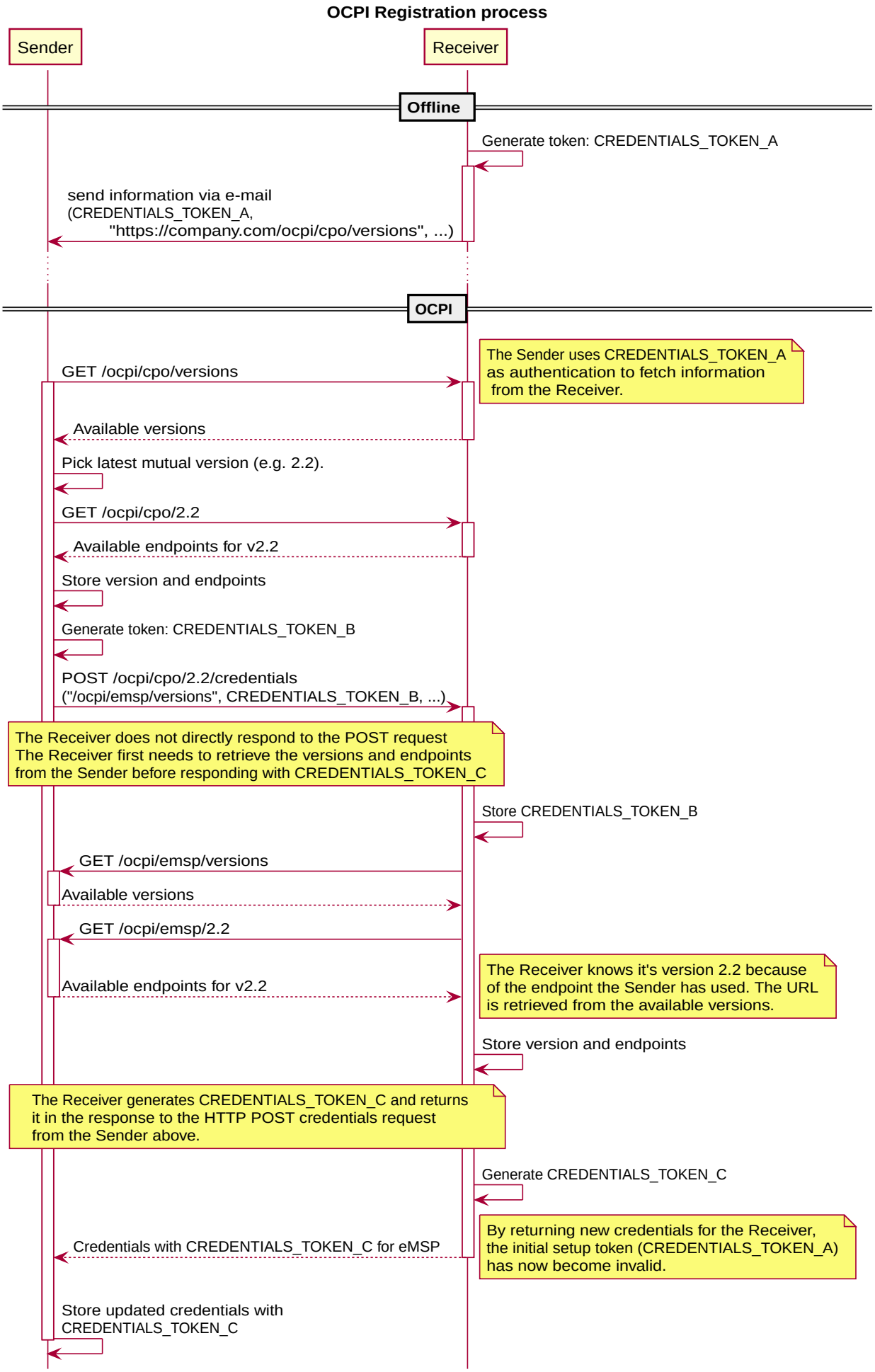


Figure 22. The OCPI registration process

Due to its symmetric nature of the credentials module, any platform can be Sender and or the Receiver for this module.

7.1.2. Updating to a newer version

At some point, both platforms will have implemented a newer OCPI version. To start using the newer version, one platform has to send a PUT request to the credentials endpoint of the other platform.

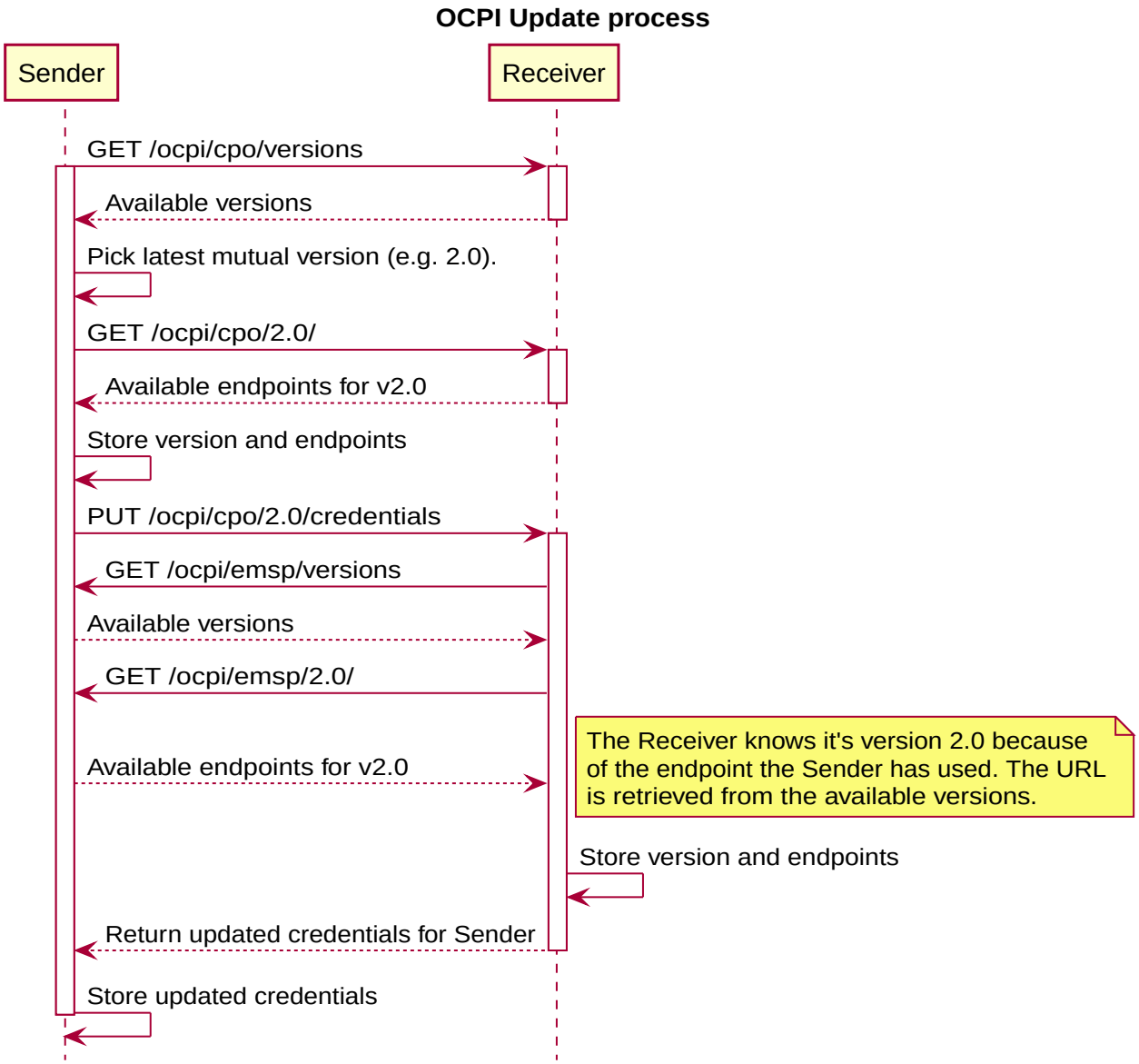


Figure 23. The OCPI update process

7.1.3. Changing endpoints for the current version

This can be done by following the update procedure for the same version.

By sending a PUT request to the credentials endpoint of this version, the other platform will fetch and store the corresponding set of endpoints.

7.1.4. Updating the credentials and resetting the credentials token

The credentials (or parts thereof, such as the credentials token) can be updated by sending the new credentials via a

PUT request to the credentials endpoint of the current version, similar to the update procedure described above.

Security advices: When one of the connecting platforms suspects that a credentials token is compromised, that platform SHALL initiate a credentials token update as soon as possible. It is advisable to renew the credentials tokens at least once a month, in case it was not detected that the credentials where compromised.

7.1.5. Errors during registration

When the server connects back to the client during the credentials registration, it might encounter problems. When this happens, the server should add the status code 3001 in the response to the POST from the client.

7.1.6. Required endpoints not available

When two platforms connect, it might happen that one of the platforms expects a certain endpoint to be available at the other platform.

For example: a Platform with a CPO role could only want to connect when the CDRs endpoint is available in an platform with an eMSP role.

In case the Sender (starting the credentials exchange process) cannot find the endpoints it expects, it is expected NOT to send the POST request with credentials to the Receiver. Log a message/notify the administrator to contact the administrator of the Receiver platform.

In case the Receiver platform that cannot find the endpoints it expects, then it is expected to respond to the request with the status code 3003.

7.2. Interfaces and endpoints

The Credentials module is different from all other OCPI modules. This module is symmetric, it has to be implemented by all OCPI implementations, and all implementations need to be able call this module on any other platform, and have to be able the handle receiving the request from another party.

Example: `/ocpi/2.2.1/credentials` and `/ocpi/emsp/2.2.1/credentials`

Method	Description
GET	Retrieves the credentials object to access the server's platform.
POST	Provides the server with a credentials object to access the client's system (i.e. register).
PUT	Provides the server with an updated credentials object to access the client's system.
PATCH	n/a
DELETE	Informs the server that its credentials to the client's system are now invalid (i.e. unregister).

7.2.1. GET Method

Retrieves the credentials object to access the server's platform. The request body is empty, the response contains the credentials object to access the server's platform. This credentials object also contains extra information about the server such as its business details.

7.2.2. POST Method

Provides the server with credentials to access the client's system. This credentials object also contains extra information about the client such as its business details.

A `POST` initiates the registration process for this endpoint's version. The server must also fetch the client's endpoints for this version.

If successful, the server must generate a new credentials token and respond with the client's new credentials to access the server's system. The credentials object in the response also contains extra information about the server such as its business details.

This method MUST return a `HTTP status code 405: method not allowed` if the client has already been registered before.

7.2.3. PUT Method

Provides the server with updated credentials to access the client's system. This credentials object also contains extra information about the client such as its business details.

A `PUT` will switch to the version that contains this credentials endpoint if it's different from the current version. The server must fetch the client's endpoints again, even if the version has not changed.

If successful, the server must generate a new credentials token for the client and respond with the client's updated credentials to access the server's system. The credentials object in the response also contains extra information about the server such as its business details.

This method MUST return a `HTTP status code 405: method not allowed` if the client has not been registered yet.

7.2.4. DELETE Method

Informs the server that its credentials to access the client's system are now invalid and can no longer be used. Both parties must end any automated communication. This is the unregistration process.

This method MUST return a `HTTP status code 405: method not allowed` if the client has not been registered before.

7.3. Object description

7.3.1. Credentials object

Property	Type	Card.	Description
token	string(64)	1	The credentials token for the other party to authenticate in your system. It should only contain printable non-whitespace ASCII characters, that is, characters with Unicode code points from the range of U+0021 up to and including U+007E.
url	URL	1	The URL to your API versions endpoint.

Property	Type	Card.	Description
hub_party_id	CiString(5)	?	The Hub party of this platform. The two-letter country code and three-character party ID are concatenated together in this field as one five-character string.
roles	CredentialsRole	+	List of the roles this platform provides.

NOTE In OCPI 2.3.0, unlike in OCPI 2.2 or 2.2.1, Roaming Hubs' platforms are expected to include the parties that are reachable through the Roaming Hub in the list in `roles`.

Every role needs a unique combination of: `role`, `party_id` and `country_code`.

A platform can have the same role more than once, each with its own unique `party_id` and `country_code`, for example when a CPO provides 'white-label' services for 'virtual' CPOs.

One or more roles and thus `party_id` and `country_code` sets are provided here to inform a server about the `party_id` and `country_code` sets a client will use when pushing Client Owned Objects. This helps a server to determine the URLs a client will use when pushing a Client Owned Object. The `country_code` is added to make certain the URL used when pushing a Client Owned Object is unique as there might be multiple parties in the world with the same `party_id`. The combination of `country_code` and `party_id` should always be unique though. A party operating in multiple countries can always use the home country of the company for all connections.

For example: EVSE IDs can be pushed under the country and provider identification of a company, even if the EVSEs are actually located in a different country. This way it is not necessary to establish one OCPI connection per country a company operates in.

The `party_id` and `country_code` given here have no direct link with the eMI3/IDACS format EVSE IDs and Contract IDs that might be used in the different OCPI modules. A party implementing OCPI MAY push EVSE IDs with an eMI3/IDACS `spot operator` different from the OCPI `party_id` and/or the `country_code`.

A Platform that supports Hub functionality with the Message routing headers SHALL give the country code and party ID of the Hub in the `hub_party_id` field.

7.3.2. Examples

Example of a minimal CPO credentials object:

```
{
    "token": "ebf3b399-779f-4497-9b9d-ac6ad3cc44d2",
    "url": "https://example.com/ocpi/versions",
    "roles": [{
        "role": "CPO",
        "party_id": "EXA",
        "country_code": "NL",
        "business_details": {
            "name": "Example Operator"
        }
    }]
}
```

Example of a combined CPO/eMSP credentials object:

```
{
    "token": "9e80a9c4-28be-11e9-b210-d663bd873d93",
    "url": "https://ocpi.example.com/versions",
    "roles": [{
        "role": "CPO",
        "party_id": "EXA",
        "country_code": "NL",
        "business_details": {
            "name": "Example Operator"
        }
    }, {
        "role": "EMSP",
        "party_id": "EXA",
        "country_code": "NL",
        "business_details": {
            "name": "Example Provider"
        }
    }]
}
```

Example of a CPO credentials object with full business details:

```
{
    "token": "9e80ae10-28be-11e9-b210-d663bd873d93",
    "url": "https://example.com/ocpi/versions",
    "roles": [{
        "role": "CPO",
        "party_id": "EXA",
        "country_code": "NL",
        "business_details": {
            "name": "Example Operator",
            "logo": {
                "url": "https://example.com/img/logo.jpg",
                "thumbnail": "https://example.com/img/logo_thumb.jpg",
                "category": "OPERATOR",
                "type": "jpeg",
                "width": 512,
                "height": 512
            },
            "website": "http://example.com"
        }
    }]
}
```

Example of a CPO credentials object for a platform that provides services for 3 CPOs:

```
{
    "token": "9e80aca8-28be-11e9-b210-d663bd873d93",
    "url": "https://ocpi.example.com/versions",
    "roles": [{
        "role": "CPO",
        "party_id": "EXO",
        "country_code": "NL",
        "business_details": {
            "name": "Excellent Operator"
        }
    }, {
        "role": "CPO",
        "party_id": "PFC",
        "country_code": "NL",
        "business_details": {
            "name": "Plug Flex Charging"
        }
    }, {
```

```
        "role": "CPO",
        "party_id": "CGP",
        "country_code": "NL",
        "business_details": {
            "name": "Charging Green Power"
        }
    }]
}
```

7.4. Data types

7.4.1. CredentialsRole *class*

Property	Type	Card.	Description
role	Role	1	Type of role.
business_details	BusinessDetails	1	Details of this party.
party_id	CiString(3)	1	CPO, eMSP (or other role) ID of this party (following the ISO-15118 standard).
country_code	CiString(2)	1	ISO-3166 alpha-2 country code of the country this party is operating in.

8. *Locations* module

Module Identifier: `locations`

Data owner: `CPO`

Type: Functional Module

The Location objects live in the CPO back-end system. They describe the charging locations of an operator.

Module dependency: the Receiver endpoint is dependent on the Tariffs module

8.1. Flow and Lifecycle

The Locations module has the Location as base object. Each Location can have multiple EVSEs (1:n) and each EVSE can have multiple Connectors (1:n). With the methods in the Receiver interface, Location data and status information can be shared with for example an eMSP and NSP. Updates can be made to a whole Location, but also only to an EVSE or a single Connector.

When a CPO creates Location objects, it pushes them to connected eMSP by calling PUT on the Receivers Locations endpoint. eMSPs who do not support Push mode need to call GET on the CPOs Locations endpoint to receive the new object. This should be done regularly to stay up to date with the CPOs data, but not too often in order to keep the load low.

If the CPO wants to replace a Location related object, they again push it to the eMSP systems by calling PUT on their Locations endpoint.

Any changes to a Location related object can also be pushed to connected eMSPs by calling the PATCH method on the eMSPs Locations endpoint, but using PATCH mode, only actual changes should be pushed. Providers who do not support Push mode need to call GET on the CPOs Locations endpoint to receive the updates.

When the CPO wants to delete an EVSE from the list of active EVSEs, they MUST update the EVSE's `status` field to `REMOVED` and call the PUT or PATCH on the eMSP system. A Location without any valid EVSE object can be considered expired and should no longer be displayed. There is no way to entirely delete Locations, EVSEs and Connectors as there are other modules like `sessions` that depend on them. If it was possible to remove these objects, those links would no longer work.

When the CPO is not sure about the state or existence of a Location, EVSE or Connector object in the eMSP's system, the CPO can perform a GET request to validate the object in the eMSP's system.

Private charging Locations, that are not to be used for public charging, SHALL NOT be published via OCPI.

8.1.1. No public charging or roaming

When a Location is not available for either Public Charging or Roaming, it is RECOMMENDED to NOT send that Location via OCPI to receiving parties.

8.1.2. Group of Charge Points

OCPP 2.0 supports a 3-tier model:

- Highest level is a Charge Point
- A Charge Point can have one or more EVSEs.
- Every EVSE can have one or more Connectors.

OCPI does not have this model:

- OCPI has Location at the highest level.
- Each location can have multiple EVSE
- Every EVSE can have one or more Connectors.

When mapping OCPP Charge Points to OCPI, there are 2 options:

- One Location for a group of Charge Points at the same location. (preferred)
- One Location per Charge Point at the same location.

OCPI prefers the first method. An EV driver does not care if a Location consists of one Charge Point with a very large amount of EVSEs, or a large amount of Charge Points with only one EVSE. The EV driver wants to know how many EVSEs are available. Grouping Charge Points in the same location into one OCPI Location will show better on a map that shows Charging Locations.

NOTE	By definition, an EVSE can only charge one EV at a time.

8.1.3. OCPP 1.x Charge Points with multiple connectors per EVSE

OCPP 1.x was not designed to support the 3-tier model. It had no notion of EVSEs. The Open Charge Alliance has written an Application Note: "Multiple Connectors per EVSE in a OCPP 1.x implementation"

The workaround:

- Define one 'virtual' EVSE per Connector.
- When a connector of an hardware EVSE becomes unavailable, set all 'virtual' EVSEs for all the connectors of the hardware EVSE to unavailable. etc.

8.2. Interfaces and endpoints

There are both, a Sender and a Receiver interface for Locations. It is advised to use the Push direction from Sender to Receiver during normal operation in order to keep the latency of updates low. The Sender interface is meant to be used when the connection between two parties is established for the first time, to retrieve the current list of Location objects with the current status, and when the Receiver is not 100% sure the Location cache is entirely up-to-date (i.e. to perform a full sync). The Receiver can also use the Sender GET Object interface to retrieve a specific Location, EVSE or Connector. This feature might be used by an Receiver that wants information about a specific Location, but has not implemented the Receiver Locations interface (i.e. cannot receive Push).

8.2.1. Sender Interface

Typically implemented by market roles like: CPO.

Method	Description
GET	Fetch a list of Locations, last updated between the {date_from} and {date_to} (paginated), or get a specific Location, EVSE or Connector.
POST	n/a
PUT	n/a
PATCH	n/a
DELETE	n/a

8.2.1.1. GET Method

Depending on the URL Segments provided, the GET request can either be used to retrieve information about a list of available Locations (with EVSEs and Connectors) at a CPO (GET List) or it can be used to retrieve information about one specific Location, EVSE or Connector (GET Object).

GET List: Request Parameters

Endpoint structure definition:

`{locations_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/cpo/2.2.1/locations/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/locations/?offset=50`

`https://www.server.com/ocpi/2.2.1/locations/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/cpo/2.2.1/locations/?offset=50&limit=100`

If the optional parameters `{date_from}` and/or `{date_to}` are provided, only Locations with (`last_updated`) between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned. In order for this to work properly, the following logic MUST be implemented accordingly: If an EVSE is updated, also the 'parent' Location's `last_updated` field needs to be updated. Similarly, if a Connector is updated, the EVSE's `last_updated` and the Location's `last_updated` fields need to be updated.

This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	no	Only return Locations that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	no	Only return Locations that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

GET List: Response Data

This endpoint returns a list of Location objects. The header will contain the pagination related headers.

Each object must contain all required fields. Fields that are not specified may be considered as null values. Any old information that is not specified in the response is considered no longer valid. For requests that use pagination, the response data provided by all the pages together is the new truth. Any old information not contained in any of the pages needs to be considered no longer valid.

Type	Card.	Description
Location	*	List of all Locations with valid EVSEs.

GET Object: Request Parameters

Endpoint structure definition for retrieving a Location, EVSE or Connector:

```
{locations_endpoint_url}/{location_id}[/{evse_uid}][/{connector_id}]
```

Examples:

```
https://www.server.com/ocpi/cpo/2.2.1/locations/LOC1
```

```
https://www.server.com/ocpi/cpo/2.2.1/locations/LOC1/3256
```

```
https://www.server.com/ocpi/cpo/2.2.1/locations/LOC1/3256/1
```

The following parameters can be provided as URL segments in the same order.

Parameter	Datatype	Required	Description
location_id	CiString(36)	yes	Location.id of the Location object to retrieve.
evse_uid	CiString(36)	no	Evse.uid, required when requesting an EVSE or Connector object.
connector_id	CiString(36)	no	Connector.id, required when requesting a Connector object.

GET Object: Response Data

The response contains the requested object.

Type	Card.	Description
Choice: one of three		
> Location	1	If a Location object was requested: the Location object.
> EVSE	1	If an EVSE object was requested: the EVSE object.
> Connector	1	If a Connector object was requested: the Connector object.

8.2.2. Receiver Interface

Typically implemented by market roles like: eMSP and NSP.

Locations are Client Owned Objects, so the end-points need to contain the required extra fields: {party_id} and {country_code}.

Endpoint structure definition:

`{locations_endpoint_url}/{country_code}/{party_id}/{location_id}[/{evse_uid}][/{connector_id}]`

Examples:

`https://www.server.com/ocpi/emsp/2.2.1/locations/BE/BEC/LOC1`

`https://server.com/ocpi/2.2.1/locations/BE/BEC/LOC1/3256`

`https://ocpi.server.com/2.2.1/locations/BE/BEC/LOC1/3256/1`

Method	Description
GET	Retrieve a Location as it is stored in the eMSP system.
POST	n/a *(use PUT)*
PUT	Push new/updated Location, EVSE and/or Connector to the eMSP.
PATCH	Notify the eMSP of partial updates to a Location, EVSE or Connector (such as the status).
DELETE	n/a *(use PATCH to update the* `status` *to* `REMOVED` *as described in Flow and Lifecycle)*

8.2.2.1. GET Method

If the CPO wants to check the status of a Location, EVSE or Connector object in the eMSP system, it might GET the object from the eMSP system for validation purposes. The CPO is the owner of the objects, so it would be illogical if the eMSP system had a different status or was missing an object. If a discrepancy is found, the CPO might push an update to the eMSP via a PUT or PATCH call.

Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO requesting data from the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting data from the eMSP system.
location_id	CiString(36)	yes	Location.id of the Location object to retrieve.
evse_uid	CiString(36)	no	Evse.uid, required when requesting an EVSE or Connector object.
connector_id	CiString(36)	no	Connector.id, required when requesting a Connector object.

Response Data

The response contains the requested object.

Type	Card.	Description
Choice: one of three		
> Location	1	If a Location object was requested: the Location object.
> EVSE	1	If an EVSE object was requested: the EVSE object.
> Connector	1	If a Connector object was requested: the Connector object.

8.2.2.2. PUT Method

The CPO pushes available Location, EVSE or Connector objects to the eMSP. PUT can be used to send new Location objects to the eMSP but also to replace existing Locations.

When the PUT only contains a Connector Object, the Receiver SHALL also set the new `last_updated` value on the parent EVSE and Location Objects.

When the PUT only contains a EVSE Object, the Receiver SHALL also set the new `last_updated` value on the parent Location Object.

Request Parameters

This is an information Push message, the objects pushed will not be owned by the eMSP. To make distinctions between objects being pushed to an eMSP from different CPOs, the {party_id} and {country_code} have to be included in the URL (as URL segments, as described in the Receiver Interface).

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO requesting this PUT to the eMSP system. This SHALL be the same value as the `country_code` in the Location object being pushed.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system. This SHALL be the same value as the `party_id` in the Location object being pushed.
location_id	CiString(36)	yes	Location.id of the new Location object, or the Location of which an EVSE or Connector object is pushed.
evse_uid	CiString(36)	no	Evse.uid, required when an EVSE or Connector object is pushed.
connector_id	CiString(36)	no	Connector.id, required when a Connector object is pushed.

Request Body

The request body contains the new/updated object.

When the PUT contains a Connector Object, the Receiver SHALL also set the new `last_updated` value on the parent EVSE and Location Objects.

When the PUT contains a EVSE Object, the Receiver SHALL also set the new `last_updated` value on the parent Location Object.

Type	Card.	Description
Choice: one of three		
> Location	1	New Location object, or Location object to replace.
> EVSE	1	New EVSE object, or EVSE object to replace.
> Connector	1	New Connector object, or Connector object to replace.

Example: add an EVSE

To add an *EVSE*, simply put the full object in an update message, including all its required fields. Since the id will be new to the eMSP's system, the receiving party will know that it is a new object. When not all required fields are specified, the object may be discarded.

```
PUT To URL: https://www.server.com/ocpi/emsp/2.2.1/locations/NL/TNM/1012/3256

{
  "uid": "3256",
  "evse_id": "BE*BEC*E041503003",
  "status": "AVAILABLE",
  "capabilities": ["RESERVABLE"],
  "connectors": [
    {
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "tariff_ids": ["14"]
    }
  ],
  "floor": -1,
  "physical_reference": 3,
  "last_updated": "2019-06-24T12:39:09Z"
}
```

8.2.2.3. PATCH Method

Same as the PUT method, but only the fields/objects that have to be updated have to be present. Other fields/objects that are not specified as part of the request are considered unchanged. Therefore, this method is not suitable to remove information shared earlier.

Any request to the PATCH method SHALL contain the `last_updated` field.

When the PATCH is on a Connector Object, the Receiver SHALL also set the new `last_updated` value on the parent EVSE and Location Objects.

When the PATCH is on a EVSE Object, the Receiver SHALL also set the new `last_updated` value on the parent Location Object.

Example: a simple status update

This is the most common type of update message. It is used to notify eMSPs that the status of an EVSE changed. In

this case it is the EVSE with uid 3255 of the Location with id 1012.

```
PATCH To URL: https://www.server.com/ocpi/emsp/2.2.1/locations/NL/TNM/1012/3255

{
  "status": "CHARGING",
  "last_updated": "2019-06-24T12:39:09Z"
}
```

Example: change the location name

In this example the name of the Location with id 1012 is being updated.

```
PATCH To URL: https://www.server.com/ocpi/emsp/2.2.1/locations/NL/TNM/1012


{
  "name": "Interparking Gent Zuid",
  "last_updated": "2019-06-24T12:39:09Z"
}
```

Example: set tariff update

In this example Connector 2 of EVSE 1 of Location 1012 receives a new pricing scheme.

```
PATCH To URL: https://www.server.com/ocpi/emsp/2.2.1/locations/NL/TNM/1012/3255/2

{
  "tariff_ids": ["15"],
  "last_updated": "2019-06-24T12:39:09Z"
}
```

Example: delete an EVSE

An EVSE can be deleted by updating its `status` property.

```
PATCH To URL: https://www.server.com/ocpi/emsp/2.2.1/locations/NL/TNM/1012/3256

{
  "status": "REMOVED",
  "last_updated": "2019-06-24T12:39:09Z"
}
```

NOTE	To inform eMSPs that an EVSE is scheduled for removal, the status_schedule field can be used._

8.3. Object description

Location, EVSE, Connector and Parking have the following relation:

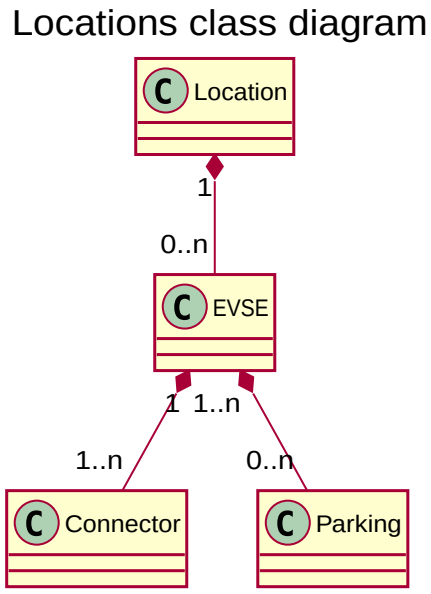


Figure 24. Location class diagram

8.3.1. *Location* Object

The *Location* object describes the location and its properties where a group of EVSEs that belong together are installed. Typically, the *Location* object is the exact location of the group of EVSEs, but it can also be the entrance of a parking garage which contains these EVSEs. The exact way to reach each EVSE can be further specified by its own properties.

Locations may be shown in apps or on websites etc. when the flag: `publish` is set to `true`. Locations that have this flag set to `false` SHALL not be shown in an app or on a website etc. unless it is to the owner of a Token in the `publish_allowed_to` list. Even parties like NSP or eMSP that do not 'own' this Token MAY show this location on an app or website, but only to the owner of that Token. If the user of their app/website has provided information about his/her Token, And that information matches all the fields of one of the PublishToken tokens in the list, then they are allowed to show this location to their user. It is not allowed in OCPI to use a Token that is not 'owned' by the eMSP itself to start a charging session.

Property	Type	Card.	Description
country_code	CiString(2)	1	ISO-3166 alpha-2 country code of the CPO that 'owns' this Location.
party_id	CiString(3)	1	ID of the CPO that 'owns' this Location (following the ISO-15118 standard).
id	CiString(36)	1	Uniquely identifies the location within the CPOs platform (and suboperator platforms). This field can never be changed, modified or renamed.

Property	Type	Card.	Description
publish	boolean	1	Defines if a Location may be published on an website or app etc. When this is set to `false`, only tokens identified in the field: `publish_allowed_to` are allowed to be shown this Location. When the same location has EVSEs that may be published and may not be published, two 'Locations' should be created.
publish_allowed_to	PublishTokenType	*	This field may only be used when the `publish` field is set to `false`. Only owners of Tokens that match all the set fields of one PublishToken in the list are allowed to be shown this location.
name	string(255)	?	Display name of the location.
address	string(45)	1	Street/block name and house number if available.
city	string(45)	1	City or town.
postal_code	string(10)	?	Postal code of the location, may only be omitted when the location has no postal code: in some countries charging locations at highways don't have postal codes.
state	string(20)	?	State or province of the location, only to be used when relevant.
country	string(3)	1	ISO 3166-1 alpha-3 code for the country of this location.
coordinates	GeoLocation	1	Coordinates of the location.
related_locations	AdditionalGeoLocation	*	Geographical location of related points relevant to the user.
parking_type	ParkingType	?	The general type of parking at the charge point location.
evses	EVSE	*	List of EVSEs that belong to this Location.
parking_places	Parking	*	List of parking places that can be used by vehicles charging at this Location.
directions	DisplayText	*	Human-readable directions on how to reach the location.
operator	BusinessDetails	?	Information of the operator. When not specified, the information retrieved from the Credentials module, selected by the `country_code` and `party_id` of this Location, should be used instead.
suboperator	BusinessDetails	?	Information of the suboperator if available.
owner	BusinessDetails	?	Information of the owner if available.

Property	Type	Card.	Description
facilities	Facility	*	Optional list of facilities this charging location directly belongs to.
time_zone	string(255)	1	One of IANA tzdata's TZ-values representing the time zone of the location. Examples: "Europe/Oslo", "Europe/Zurich". (http://www.iana.org/time-zones)
opening_times	Hours	?	The times when the EVSEs at the location can be accessed for charging.
charging_when_closed	boolean	?	Indicates if the EVSEs are still charging outside the opening hours of the location. E.g. when the parking garage closes its barriers over night, is it allowed to charge till the next morning? Default: **true**
images	Image	*	Links to images related to the location such as photos or logos.
energy_mix	EnergyMix	?	Details on the energy supplied at this location.
help_phone	CiString(25)	?	A telephone number that a Driver using the Location may call for assistance. Calling this number will typically connect the caller to the CPO's customer service department.
last_updated	DateTime	1	Timestamp when this Location or one of its EVSEs or Connectors were last updated (or created).

Private Charge Points, home or business that do not need to be published on apps, and do not require remote control via OCPI, SHOULD not be PUT via the OCPI Locations module.

8.3.1.1. Example public charging location

This is an example of a public charging location. Can be used by any EV Driver as long as his eMSP has a roaming agreement with the CPO. Or the Charge Point has an ad-hoc payment possibility

- `publish` = `true`
- `parking_type` = `ON_STREET` but could also be another value.
- `EVSE.parking_restrictions` not used.


```
{
  "country_code": "BE",
  "party_id": "BEC",
  "id": "LOC1",
  "publish": true,
  "name": "Gent Zuid",
  "address": "F.Rooseveltlaan 3A",
  "city": "Gent",
  "postal_code": "9000",
  "country": "BEL",
  "coordinates": {
    "latitude": "51.047599",
    "longitude": "3.729944"
  },
```

```
  "parking_type": "PARKING_GARAGE",
  "evses": [{
    "uid": "3256",
    "evse_id": "BE*BEC*E041503001",
    "status": "AVAILABLE",
    "capabilities": [
      "RESERVABLE"
    ],
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "CABLE",
      "power_type": "AC_3_PHASE",
      "max_voltage": 220,
      "max_amperage": 16,
      "tariff_ids": ["11"],
      "last_updated": "2015-03-16T10:10:02Z"
    }, {
      "id": "2",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "max_voltage": 220,
      "max_amperage": 16,
      "tariff_ids": ["13"],
      "last_updated": "2015-03-18T08:12:01Z"
    }],
    "parking": ["1", "2"],
    "physical_reference": "1",
    "floor_level": "-1",
    "last_updated": "2015-06-28T08:12:01Z"
  }, {
    "uid": "3257",
    "evse_id": "BE*BEC*E041503002",
    "status": "RESERVED",
    "capabilities": [
      "RESERVABLE"
    ],
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "max_voltage": 220,
      "max_amperage": 16,
      "tariff_ids": ["12"],
      "last_updated": "2015-06-29T20:39:09Z"
    }],
    "parking": ["2", "3"],
    "physical_reference": "2",
    "floor_level": "-2",
    "last_updated": "2015-06-29T20:39:09Z"
  }],
  "parking_places": [{
      "id": "1",
      "physical_reference": "B1",
      "vehicle_types": ["PERSONAL_VEHICLE"],
      "reservation_required": false,
      "restricted_to_type": true
    }, {
      "id": "2",
      "physical_reference": "B2",
      "vehicle_types": ["PERSONAL_VEHICLE"],
      "reservation_required": false,
      "restricted_to_type": true
    },
    {
      "id": "3",
      "physical_reference": "B3",
      "vehicle_types": ["PERSONAL_VEHICLE"],
```

```
      "reservation_required": false,
      "restricted_to_type": true
    }
  ],
  "operator": {
    "name": "BeCharged"
  },
  "time_zone": "Europe/Brussels",
  "last_updated": "2015-06-29T20:39:09Z"
}
```

8.3.1.2. Example destination charging location

This is an example of a destination charging location. This is a Location where only guests, employees or customers can charge. For an EV driver, it can be useful to know if he/she can charge at his destination.

For example at a restaurant, only customers of the restaurant can charge their EV. Or at an office building where employees and guest of the office can charge their EV.

Locations you can think of where this is useful: restaurants, bars, clubs, theme parks, stores, supermarkets, company building, office buildings, etc.

- `publish` = `true`
- `parking_type` = `PARKING_LOT` (but could also be `PARKING_GARAGE`, `ON_DRIVEWAY` or `UNDERGROUND_GARAGE`)
- `EVSE.parking_restrictions` = `CUSTOMERS`

```
{
  "country_code": "NL",
  "party_id": "ALF",
  "id": "3e7b39c2-10d0-4138-a8b3-8509a25f9920",
  "publish": true,
  "name": "ihomer",
  "address": "Tamboerijn 7",
  "city": "Etten-Leur",
  "postal_code": "4876 BS",
  "country": "NLD",
  "coordinates": {
    "latitude": "51.562787",
    "longitude": "4.638975"
  },
  "parking_type": "PARKING_LOT",
  "evses": [{
    "uid": "fd855359-bc81-47bb-bb89-849ae3dac89e",
    "evse_id": "NL*ALF*E000000001",
    "status": "AVAILABLE",
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "max_voltage": 220,
      "max_amperage": 16,
      "last_updated": "2019-07-01T12:12:11Z"
    }],
    "parking_restrictions": [ "CUSTOMERS" ],
    "last_updated": "2019-07-01T12:12:11Z"
  }],
  "time_zone": "Europe/Amsterdam",
  "last_updated": "2019-07-01T12:12:11Z"
}
```

8.3.1.3. Example destination charging location not published, but paid guest usage possible

This is an example of a destination charging location. But the owner of the location has requested not to publish the location in Apps or on websites.

Charging is still possible: EV drivers of an eMSP with a roaming agreement can still charge their EV. The eMSP helpdesk can use the information from the Location module to help the driver, maybe even start a session for a driver. Starting a session from an App is not possible, because the driver will not be able to select the Charge Point on a map.

In case the EV driver is not billed for charging, there is, in such a case, no reason to publish the location via OCPI.

- `publish` = `false`
- `publish_allowed_to` not used
- `parking_type` = not used `
- `EVSE.parking_restrictions` = `CUSTOMERS` may still be useful so a support desk can also tell this to a customer.


```
{
  "country_code": "NL",
  "party_id": "ALF",
  "id": "3e7b39c2-10d0-4138-a8b3-8509a25f9920",
  "publish": false,
  "name": "ihomer",
  "address": "Tamboerijn 7",
  "city": "Etten-Leur",
  "postal_code": "4876 BS",
  "country": "NLD",
  "coordinates": {
    "latitude": "51.562787",
    "longitude": "4.638975"
  },
  "evses": [{
    "uid": "fd855359-bc81-47bb-bb89-849ae3dac89e",
    "evse_id": "NL*ALF*E000000001",
    "status": "AVAILABLE",
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "max_voltage": 220,
      "max_amperage": 16,
      "last_updated": "2019-07-01T12:12:11Z"
    }],
    "parking_restrictions": [ "CUSTOMERS" ],
    "last_updated": "2019-07-01T12:12:11Z"
  }],
  "time_zone": "Europe/Amsterdam",
  "last_updated": "2019-07-01T12:12:11Z"
}
```


8.3.1.4. Example charging location with limited visibility

This is an example of a charging location that only a limited group can see (and use) via an App or website.

Typical examples where this is useful:

- Charge Points in the parking garage of an apartment building. Only owners can see/control the Charge Points.

- Charge Points at an office, for employees only. Only employees can see/control the Charge Points.
- Charge Points at vehicle depot. Any employee can see/control an charge point, even transaction they did not start. Use `group_id` for this.

The locations SHALL NOT be published to the general public. Only selected Tokens can see (and control) the Charge Points via eMSP app.

- `publish` = `false`
- `publish_allowed_to` contains list with information of Tokens that are allowed to be shown the `Location`.
- `parking_type` = `UNDERGROUND_GARAGE` (but could also be `PARKING_GARAGE`, `ON_DRIVEWAY` or `PARKING_LOT`)


```
{
  "country_code": "NL",
  "party_id": "ALL",
  "id": "f76c2e0c-a6ef-4f67-bf23-6a187e5ca0e0",
  "publish": false,
  "publish_allowed_to": [{
    "visual_number": "12345-67",
    "issuer": "NewMotion"
  }, {
    "visual_number": "0055375624",
    "issuer": "ANWB"
  }, {
    "uid": "12345678905880",
    "type": "RFID"
  }],
  "name": "Water State",
  "address": "Taco van der Veenplein 12",
  "city": "Leeuwarden",
  "postal_code": "8923 EM",
  "country": "NLD",
  "coordinates": {
    "latitude": "53.213763",
    "longitude": "5.804638"
  },
  "parking_type": "UNDERGROUND_GARAGE",
  "evses": [{
    "uid": "8c1b3487-61ac-40a7-a367-21eee99dbd90",
    "evse_id": "NL*ALL*EGO0000013",
    "status": "AVAILABLE",
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "max_voltage": 230,
      "max_amperage": 16,
      "last_updated": "2019-09-27T00:19:45Z"
    }],
    "last_updated": "2019-09-27T00:19:45Z"
  }],
  "time_zone": "Europe/Amsterdam",
  "last_updated": "2019-09-27T00:19:45Z"
}
```


8.3.1.5. Example private charge point with eMSP app control

This is an example of a private/home charge point that needs to be controlled via an eMSP App.

The locations SHALL NOT be published to the general public. Only the owner, identified by his/her Token can see (and control) the Charge Points via an eMSP app.

- `publish` = `false`
- `publish_allowed_to` contains the information of the Tokens of the owner.
- `parking_type` = not used, not relevant, owner knows where his Charge Point is.


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "a5295927-09b9-4a71-b4b9-a5fffdfa0b77",
  "publish": false,
  "publish_allowed_to": [{
    "visual_number": "0123456-99",
    "issuer": "MoveMove"
  }],
  "address": "Krautwigstraße 283A",
  "city": "Köln",
  "postal_code": "50931",
  "country": "DEU",
  "coordinates": {
    "latitude": "50.931826",
    "longitude": "6.964043"
  },
  "parking_type": "ON_DRIVEWAY",
  "evses": [{
    "uid": "4534ad5f-45be-428b-bfd0-fa489dda932d",
    "evse_id": "DE*ALL*EGO0000001",
    "status": "AVAILABLE",
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_1_PHASE",
      "max_voltage": 230,
      "max_amperage": 8,
      "last_updated": "2019-04-05T17:17:56Z"
    }],
    "last_updated": "2019-04-05T17:17:56Z"
  }],
  "time_zone": "Europe/Berlin",
  "last_updated": "2019-04-05T17:17:56Z"
}
```


8.3.1.6. Example charge point in a parking garage with opening hours

This is an example of a charge point, located in a parking garage with limited opening hours: 7:00 - 18:00.

If the EV is left in the parking garage overnight, the car will still be charged.

- `publish` = `true`
- `parking_type` = `PARKING_GARAGE` but could also be another value.
- `EVSE.parking_restrictions` not used.
- `opening_times` is used.
- `charging_when_closed` = `true`


```
{
  "country_code": "SE",
  "party_id": "EVC",
  "id": "cbb0df21-d17d-40ba-a4aa-dc588c8f98cb",
  "publish": true,
  "name": "P-Huset Leonard",
```

```
  "address": "Claesgatan 6",
  "city": "Malmö",
  "postal_code": "214 26",
  "country": "SWE",
  "coordinates": {
    "latitude": "55.590325",
    "longitude": "13.008307"
  },
  "parking_type": "PARKING_GARAGE",
  "evses": [{
    "uid": "eccb8dd9-4189-433e-b100-cc0945dd17dc",
    "evse_id": "SE*EVC*E000000123",
    "status": "AVAILABLE",
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "max_voltage": 230,
      "max_amperage": 32,
      "last_updated": "2017-03-07T02:21:22Z"
    }],
    "parking": ["000e3877-87bf-473e-8e71-70d3aa6d64ea"],
    "last_updated": "2017-03-07T02:21:22Z"
  }],
  "parking_places": [{
    "id": "000e3877-87bf-473e-8e71-70d3aa6d64ea",
    "physical_reference": "A0",
    "vehicle_types": ["PERSONAL_VEHICLE"],
    "reservation_required": false,
    "restricted_to_type": true
  }],
  "time_zone": "Europe/Stockholm",
  "opening_times": {
    "twentyfourseven": false,
    "regular_hours": [{
      "weekday": 1,
      "period_begin": "07:00",
      "period_end": "18:00"
    }, {
      "weekday": 2,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 3,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 4,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 5,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 6,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 7,
      "period_begin": "07:00",
      "period_end": "18:00"
    }]
  },
  "charging_when_closed": true,
  "last_updated": "2017-03-07T02:21:22Z"
}
```

8.3.2. *EVSE* Object

The *EVSE* object describes the part that controls the power supply to a single EV in a single session. It always belongs to a Location object. The object only contains directions to get from the location itself to the EVSE (i.e. *floor*, *physical_reference* or *directions*).

When the directional properties of an EVSE are insufficient to reach the EVSE from the *Location* point, then it typically indicates that the EVSE should be put in a different *Location* object (sometimes with the same address but with different coordinates/directions).

An *EVSE* object has a list of Connectors which can not be used simultaneously: only one connector per EVSE can be used at the time.

Property	Type	Card.	Description
uid	CiString(36)	1	Uniquely identifies the EVSE within the CPOs platform (and suboperator platforms). This field can never be changed, modified or renamed. This is the 'technical' identification of the EVSE, not to be used as 'human readable' identification, use the field `evse_id` for that. This field is named `uid` instead of `id`, because `id` could be confused with `evse_id` which is a field in a specific format defined by eMI3 and IDACS. Note that in order to fulfill both the requirement that an EVSE's `uid` be unique within a CPO's platform and the requirement that EVSEs are never deleted, a CPO will typically want to avoid using identifiers of the physical hardware for this `uid` property. If they do use such a physical identifier, they will find themselves breaking the uniqueness requirement for `uid` when the same physical EVSE is redeployed at another Location.
evse_id	CiString(48)	?	Compliant with the following specification for EVSE ID: "E-mobility ID-codes: the purpose of IDs, ID usage and ID format" (https://evroaming.org/contract-evse-ids/). Optional because: if an `evse_id` is to be re-used in the real world, the `evse_id` can be removed from an EVSE object if the `status` is set to `REMOVED`.
status	Status	1	Indicates the current status of the EVSE.
status_schedule	StatusSchedule	*	Indicates a planned status update of the EVSE.
capabilities	Capability	*	List of functionalities that the EVSE is capable of.
connectors	Connector	+	List of available connectors on the EVSE.
floor_level	string(4)	?	Level on which the Charge Point is located (in garage buildings) in the locally displayed numbering scheme.
coordinates	GeoLocation	?	Coordinates of the EVSE.
physical_reference	string(16)	?	A number/string printed on the outside of the EVSE for visual identification.

Property	Type	Card.	Description
directions	DisplayText	*	Multi-language human-readable directions when more detailed information on how to reach the EVSE from the *Location* is required.
parking_restrictions	ParkingRestriction	*	All applicable restrictions on who can charge at the EVSE, apart from those related to the vehicle type.
parking	EVSEParking	*	References to the parking space or spaces that can be used by vehicles charging at this EVSE.
images	Image	*	Links to images related to the EVSE such as photos or logos.
accepted_service_providers	String[50]	*	A list of the names of the eMSPs offering contract-based payment options that are accepted at this EVSE. Note that this field is added specifically to allow European CPOs to comply with a regulatory requirement to provide this data to National Access Points (NAPs). When this requirement does not apply, this field can be left out.
last_updated	DateTime	1	Timestamp when this EVSE or one of its Connectors was last updated (or created).

NOTE

OCPP 1.x does not have good support for Charge Points that have multiple connectors per EVSE. To make `StartSession` over OCPI work, the CPO SHOULD present the different connectors of an EVSE as separate EVSE, as is also written by the OCA in the application note: "Multiple Connectors per EVSE in a OCPP 1.x implementation".

8.3.3. *Connector* Object

A *Connector* is the *socket* or *cable and plug* available for the EV to use. A single EVSE may provide multiple Connectors but only one of them can be in use at the same time. A Connector always belongs to an EVSE object.

Property	Type	Card.	Description
id	CiString(36)	1	Identifier of the Connector within the EVSE. Two Connectors may have the same id as long as they do not belong to the same *EVSE* object.
standard	ConnectorType	1	The standard of the installed connector.
format	ConnectorFormat	1	The format (socket/cable) of the installed connector.
power_type	PowerType	1	
max_voltage	int	1	Maximum voltage of the connector (line to neutral for AC_3_PHASE), in volt [V]. For example: DC Chargers might vary the voltage during charging when battery almost full.
max_amperage	int	1	Maximum amperage of the connector, in ampere [A].

Property	Type	Card.	Description
max_electric_power	int	?	Maximum electric power that can be delivered by this connector, in Watts (W). When the maximum electric power is lower than the calculated value from `voltage` and `amperage`, this value should be set. For example: A DC Charge Point which can delivers up to 920V and up to 400A can be limited to a maximum of 150kW (max_electric_power = 150000). Depending on the car, it may supply max voltage or current, but not both at the same time. For AC Charge Points, the amount of phases used can also have influence on the maximum power.
tariff_ids	CiString(36)	*	Identifiers of the currently valid charging tariffs. Multiple tariffs are possible, but only one of each Tariff.type can be active at the same time. Tariffs with the same type are only allowed if they are not active at the same time: start_date_time and end_date_time period not overlapping. When preference-based smart charging is supported, one tariff for every possible ProfileType should be provided. These tell the user about the options they have at this Connector, and what the tariff is for every option. For a "free of charge" tariff, this field should be set and point to a defined "free of charge" tariff.
terms_and_conditions	URL	?	URL to the operator's terms and conditions.
capabilities	ConnectorCapability	*	A list of functionalities that the connector is capable of.
last_updated	DateTime	1	Timestamp when this Connector was last updated (or created).

8.3.4. *Parking* object

Describes a parking space that a vehicle can be parked in while charging.

For EVSEs around which no identifiable delineated parking spaces are available, a Parking object may describe the limitations that apply for parking near the EVSE without describing a specific space. This occurs a lot with streetside parking, for example.

NOTE

Parking objects were newly added in OCPI 2.3.0 relative to OCPI 2.2.1. The purpose of Parking objects is to allow CPOs in the EU to comply with requirements in the EU's Alternative Fuel Infrasturcture Regulation (AFIR) which requires CPOs to report the number of parking spots and certain properties of those parking spots to NAPs. When CPOs are not talking to NAPs, or not under EU jurisdiction, they are free to not send Parking objects in their Locations. All Locations receivers who are not NAPs are free to ignore Parking objects in the Location data that they receive.

Property	Type	Card.	Description
id	CiString[36]	1	The identifier for this parking space. The value of this field MUST be unique among all Parking objects in the same Location object.
physical_reference	String[12]	?	A string identifier for the parking place that is physically visible on-site to drivers using the parking space. This could be a short identifier painted on the surface of a parking place in a parking garage for example.
vehicle_types	VehicleType	+	The vehicle types that the parking is designed to accommodate.
max_vehicle_weight	number	?	The maximum vehicle weight that can park at the EVSE, in kilograms. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
max_vehicle_height	number	?	The maximum vehicle height that can park at the EVSE, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
max_vehicle_length	number	?	The maximum vehicle length that can park at the EVSE, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
max_vehicle_width	number	?	The maximum vehicle width that can park at the EVSE, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
parking_space_length	number	?	The length of the parking space, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
parking_space_width	number	?	The width of the parking space, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
dangerous_goods_allowed	boolean	?	Whether vehicles loaded with dangerous substances are allowed to park at the EVSE. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
direction	ParkingDirection	?	The direction in which the vehicle is to be parked next to the EVSE.
drive_through	boolean	?	Whether a vehicle can stop, charge, and proceed without reversing into or out of a parking space. This should only be set to true if driving through is possible for all vehicle types listed in the `vehicle_types` field.
restricted_to_type	boolean	1	Whether it is forbidden for vehicles of a type not listed in `vehicle_types` to park at the EVSE, even if they can physically park there safely.

Property	Type	Card.	Description
reservation_required	boolean	1	Whether a reservation is required for parking at the EVSE.
time_limit	number	?	A time limit. If this field is present, vehicles may not park in this parking longer than this number of minutes.
roofed	boolean	?	Whether the vehicle will be parked under a roof while charging.
images	Image	*	Photos of the parking space at the EVSE. At least one photograph should be provided if the value of `vehicle_types` includes the `DISABLED` vehicle type.
lighting	boolean	?	Whether the parking space for the EVSE is lit by artificial lighting.
refrigeration_outlet	boolean	?	Whether a power outlet is available to power a transport truck's load refrigeration while the vehicle is parked.
standards	CiString[36]	*	A list of standards that the parking space conforms to, e.g. PAS 1899 for parking for people with disabilities.
apds_reference	CiString	?	Reference to an Alliance for Parking Data Standards (APDS) element describing this parking. The referenced element may be a Place, Space or other hierarchy element defined by APDS.

8.4. Data types

8.4.1. AdditionalGeoLocation *class*

This class defines an additional geo location that is relevant for the Charge Point. The geodetic system to be used is WGS 84.

Property	Type	Card.	Description
latitude	string(10)	1	Latitude of the point in decimal degree. Example: 50.770774. Decimal separator: "." Regex: `-?[0-9]{1,2}\.[0-9]{5,7}`
longitude	string(11)	1	Longitude of the point in decimal degree. Example: -126.104965. Decimal separator: "." Regex: `-?[0-9]{1,3}\.[0-9]{5,7}`
name	DisplayText	?	Name of the point in local language or as written at the location. For example the street name of a parking lot entrance or it's number.

8.4.2. BusinessDetails *class*

Property	Type	Card.	Description
name	string(100)	1	Name of the operator.
website	URL	?	Link to the operator's website.

Property	Type	Card.	Description
logo	Image	?	Image link to the operator's logo.

8.4.3. Capability *OpenEnum*

The capabilities of an EVSE.

Value	Description
CHARGING_PROFILE_CAPABLE	The EVSE supports charging profiles.
CHARGING_PREFERENCES_CAPABLE	The EVSE supports charging preferences.
CHIP_CARD_SUPPORT	EVSE has a payment terminal that supports chip cards.
CONTACTLESS_CARD_SUPPORT	EVSE has a payment terminal that supports contactless cards.
CREDIT_CARD_PAYABLE	EVSE has a payment terminal that makes it possible to pay for charging using a credit card.
DEBIT_CARD_PAYABLE	EVSE has a payment terminal that makes it possible to pay for charging using a debit card.
PED_TERMINAL	EVSE has a payment terminal with a pin-code entry device.
REMOTE_START_STOP_CAPABLE	The EVSE can remotely be started/stopped.
RESERVABLE	The EVSE can be reserved.
RFID_READER	Charging at this EVSE can be authorized with an RFID token.
START_SESSION_CONNECTOR_REQUIRED	When a StartSession is sent to this EVSE, the MSP is required to add the optional `connector_id` field in the StartSession object.
TOKEN_GROUP_CAPABLE	This EVSE supports token groups, two or more tokens work as one, so that a session can be started with one token and stopped with another (handy when a card and key-fob are given to the EV-driver).
UNLOCK_CAPABLE	Connectors have mechanical lock that can be requested by the eMSP to be unlocked.

When a Charge Point supports ad-hoc payments with a payment terminal, please use a combination of the following values to explain the possibilities of the terminal: CHIP_CARD_SUPPORT, CONTACTLESS_CARD_SUPPORT, CREDIT_CARD_PAYABLE, DEBIT_CARD_PAYABLE, PED_TERMINAL.

There are Charge Points in the field that do not yet support OCPP 2.x. If these Charge Points have multiple connectors per EVSE, the CPO needs to know which connector to start when receiving a StartSession for the given EVSE. If this is the case, the CPO should set the `START_SESSION_CONNECTOR_REQUIRED` capability on the given EVSE.

8.4.4. ConnectorCapability *OpenEnum*

Functionalities that a Connector may or may not support.

NOTE that these capabilities are meant to signal to eMSPs and their Drivers that a Driver can indeed use

these functionalities at a Connector. Mere support for a standard by the charging hardware is not enough to warrant the presence of these capabilities.

Value	Description
ISO_15118_2_PLUG_AND_CHARGE	The Connector supports authentication of the Driver using a contract certificate stored in the vehicle according to ISO 15118-2.
ISO_15118_20_PLUG_AND_CHARGE	The Connector supports authentication of the Driver using a contract certificate stored in the vehicle according to ISO 15118-20.

8.4.5. ConnectorFormat *enum*

The format of the connector, whether it is a socket or a plug.

Value	Description
SOCKET	The connector is a socket; the EV user needs to bring a fitting plug.
CABLE	The connector is an attached cable; the EV users car needs to have a fitting inlet.

8.4.6. ConnectorType *OpenEnum*

The socket or plug standard of the charging point.

Value	Description
CHADEMO	The connector type is CHAdeMO, DC
CHAOJI	The ChaoJi connector. The new generation charging connector, harmonized between CHAdeMO and GB/T. DC.
DOMESTIC_A	Standard/Domestic household, type "A", NEMA 1-15, 2 pins
DOMESTIC_B	Standard/Domestic household, type "B", NEMA 5-15, 3 pins
DOMESTIC_C	Standard/Domestic household, type "C", CEE 7/17, 2 pins
DOMESTIC_D	Standard/Domestic household, type "D", 3 pin
DOMESTIC_E	Standard/Domestic household, type "E", CEE 7/5 3 pins
DOMESTIC_F	Standard/Domestic household, type "F", CEE 7/4, Schuko, 3 pins
DOMESTIC_G	Standard/Domestic household, type "G", BS 1363, Commonwealth, 3 pins
DOMESTIC_H	Standard/Domestic household, type "H", SI-32, 3 pins
DOMESTIC_I	Standard/Domestic household, type "I", AS 3112, 3 pins
DOMESTIC_J	Standard/Domestic household, type "J", SEV 1011, 3 pins
DOMESTIC_K	Standard/Domestic household, type "K", DS 60884-2-D1, 3 pins
DOMESTIC_L	Standard/Domestic household, type "L", CEI 23-16-VII, 3 pins
DOMESTIC_M	Standard/Domestic household, type "M", BS 546, 3 pins

Value	Description
DOMESTIC_N	Standard/Domestic household, type "N", NBR 14136, 3 pins
DOMESTIC_O	Standard/Domestic household, type "O", TIS 166-2549, 3 pins
GBT_AC	Guobiao GB/T 20234.2 AC socket/connector
GBT_DC	Guobiao GB/T 20234.3 DC connector
IEC_60309_2_single_16	IEC 60309-2 Industrial Connector single phase 16 amperes (usually blue)
IEC_60309_2_three_16	IEC 60309-2 Industrial Connector three phases 16 amperes (usually red)
IEC_60309_2_three_32	IEC 60309-2 Industrial Connector three phases 32 amperes (usually red)
IEC_60309_2_three_64	IEC 60309-2 Industrial Connector three phases 64 amperes (usually red)
IEC_62196_T1	IEC 62196 Type 1 "SAE J1772"
IEC_62196_T1_COMBO	Combo Type 1 based, DC
IEC_62196_T2	IEC 62196 Type 2 "Mennekes"
IEC_62196_T2_COMBO	Combo Type 2 based, DC
IEC_62196_T3A	IEC 62196 Type 3A
IEC_62196_T3C	IEC 62196 Type 3C "Scame"
MCS	The MegaWatt Charging System (MCS) connector as developed by CharIN
NEMA_5_20	NEMA 5-20, 3 pins
NEMA_6_30	NEMA 6-30, 3 pins
NEMA_6_50	NEMA 6-50, 3 pins
NEMA_10_30	NEMA 10-30, 3 pins
NEMA_10_50	NEMA 10-50, 3 pins
NEMA_14_30	NEMA 14-30, 3 pins, rating of 30 A
NEMA_14_50	NEMA 14-50, 3 pins, rating of 50 A
PANTOGRAPH_BOTTOM_UP	On-board Bottom-up-Pantograph typically for bus charging
PANTOGRAPH_TOP_DOWN	Off-board Top-down-Pantograph typically for bus charging
SAE_J3400	SAE J3400, also known as North American Charging Standard (NACS), developed by Tesla, Inc in 2021.
TESLA_R	Tesla Connector "Roadster"-type (round, 4 pin)
TESLA_S	Tesla Connector "Model-S"-type (oval, 5 pin). Mechanically compatible with SAE J3400 but uses CAN bus for communication instead of power line communication.

8.4.7. EnergyMix *class*

This type is used to specify the energy mix and environmental impact of the supplied energy at a location or in a tariff.

Property	Type	Card.	Description
is_green_energy	boolean	1	True if 100% from regenerative sources. (CO2 and nuclear waste is zero)
energy_sources	EnergySource	*	Key-value pairs (enum + percentage) of energy sources of this location's tariff.
environ_impact	EnvironmentalImpact	*	Key-value pairs (enum + percentage) of nuclear waste and CO2 exhaust of this location's tariff.
supplier_name	string(64)	?	Name of the energy supplier, delivering the energy for this location or tariff.*
energy_product_name	string(64)	?	Name of the energy suppliers product/tariff plan used at this location.*

** These fields can be used to look up energy qualification or to show it directly to the customer (for well-known brands like Greenpeace Energy, etc.)*

8.4.7.1. Examples

Simple:

```
"energy_mix": {
  "is_green_energy": true
}
```

Tariff energy provider name:

```
"energy_mix": {
  "is_green_energy": true,
  "supplier_name": "Greenpeace Energy eG",
  "energy_product_name": "eco-power"
}
```

Complete:

```
"energy_mix": {
  "is_green_energy": false,
  "energy_sources": [
  { "source": "GENERAL_GREEN",  "percentage": 35.9 },
  { "source": "GAS",            "percentage": 6.3  },
  { "source": "COAL",           "percentage": 33.2 },
  { "source": "GENERAL_FOSSIL", "percentage": 2.9  },
  { "source": "NUCLEAR",        "percentage": 21.7 }
  ],
  "environ_impact": [
  { "category": "NUCLEAR_WASTE",  "amount": 0.0006   },
  { "category": "CARBON_DIOXIDE", "amount": 372      }
```

```
    ],
    "supplier_name":       "E.ON Energy Deutschland",
    "energy_product_name": "E.ON DirektStrom eco"
  }
```

8.4.8. EnergySource *class*

Key-value pairs (enum + percentage) of energy sources. All given values of all categories should add up to 100 percent.

Property	Type	Card.	Description
source	EnergySourceCategory	1	The type of energy source.
percentage	number	1	Percentage of this source (0-100) in the mix.

8.4.9. EnergySourceCategory *enum*

Categories of energy sources.

Value	Description
NUCLEAR	Nuclear power sources.
GENERAL_FOSSIL	All kinds of fossil power sources.
COAL	Fossil power from coal.
GAS	Fossil power from gas.
GENERAL_GREEN	All kinds of regenerative power sources.
SOLAR	Regenerative power from PV.
WIND	Regenerative power from wind turbines.
WATER	Regenerative power from water turbines.

8.4.10. EnvironmentalImpact *class*

Amount of waste produced/emitted per kWh.

Property	Type	Card.	Description
category	EnvironmentalImpactCategory	1	The environmental impact category of this value.
amount	number	1	Amount of this portion in g/kWh.

8.4.11. EnvironmentalImpactCategory *OpenEnum*

Categories of environmental impact values.

Value	Description
NUCLEAR_WASTE	Produced nuclear waste in grams per kilowatthour.
CARBON_DIOXIDE	Exhausted carbon dioxide in grams per kilowatthour.

8.4.12. EVSEParking *class*

A link between an EVSE and a Parking object. The presence of an EVSEParking object in an EVSE indicates that a certain parking space can be used when charging at that EVSE.

Property	Type	Card.	Description
parking_id	CiString[36]	1	The ID of the Parking. The string in this field refers to a Parking object from the containing Location's parking_places field by its `id` field.
evse_position	EVSEPosition	?	The position of the EVSE relative to the parking space.

8.4.13. EVSEPosition *enum*

The position of an EVSE relative to the EVSE's parking space.

Value	Description
LEFT	The EVSE is to the left of the vehicle. For streetside parking, the CPO can assume the vehicle is facing the same way as traffic on the side of the road that the EVSE is on. This means that LEFT is used for all streetside parking in locales with left-hand traffic. For parking spaces leading sideways from a roadway, the CPO can assume the vehicle is parking with the nose away from the roadway (that is, entering the parking space driving forward).
RIGHT	The EVSE is to the right of the vehicle when parked. For streetside parking, the CPO can assume the vehicle is facing the same way as traffic on the side of the road that the EVSE is on. This means that RIGHT is used for all streetside parking in locales with right-hand traffic. For parking spaces leading sideways from a roadway, the CPO can assume the vehicle is parking with the nose away from the roadway (that is, entering the parking space driving forward).
CENTER	The EVSE is at the center of the impassable narrow end of a parking space.

8.4.14. ExceptionalPeriod *class*

Specifies one exceptional period for opening or access hours.

Property	Type	Card.	Description
period_begin	DateTime	1	Begin of the exception. In UTC, `time_zone` field can be used to convert to local time.
period_end	DateTime	1	End of the exception. In UTC, `time_zone` field can be used to convert to local time.

8.4.15. Facility *OpenEnum*

Value	Description
HOTEL	A hotel.
RESTAURANT	A restaurant.
CAFE	A cafe.
MALL	A mall or shopping center.
SUPERMARKET	A supermarket.
SPORT	Sport facilities: gym, field etc.
RECREATION_AREA	A recreation area.
NATURE	Located in, or close to, a park, nature reserve etc.
MUSEUM	A museum.
BIKE_SHARING	A bike/e-bike/e-scooter sharing location.
BUS_STOP	A bus stop.
TAXI_STAND	A taxi stand.
TRAM_STOP	A tram stop/station.
METRO_STATION	A metro station.
TRAIN_STATION	A train station.
AIRPORT	An airport.
PARKING_LOT	A parking lot.
CARPOOL_PARKING	A carpool parking.
FUEL_STATION	A Fuel station.
WIFI	Wifi or other type of internet available.

8.4.16. GeoLocation *class*

This class defines the geo location of the Charge Point. The geodetic system to be used is WGS 84.

Property	Type	Card.	Description
latitude	string(10)	1	Latitude of the point in decimal degree. Example: 50.770774. Decimal separator: "." Regex: `-?[0-9]{1,2}\.[0-9]{5,7}`
longitude	string(11)	1	Longitude of the point in decimal degree. Example: -126.104965. Decimal separator: "." Regex: `-?[0-9]{1,3}\.[0-9]{5,7}`

NOTE Five decimal places is seen as a minimum for GPS coordinates of the Charge Point as this gives approximately 1 meter precision. More is always better. Seven decimal places gives approximately

1cm precision.

8.4.17. Hours *class*

Opening and access hours of the location.

Property	Type	Card.	Description
twentyfourseven	boolean	1	True to represent 24 hours a day and 7 days a week, except the given exceptions.
regular_hours	RegularHours	*	Regular hours, weekday-based. Only to be used if `twentyfourseven=false`, then this field needs to contain at least one RegularHours object.
exceptional_openings	ExceptionalPeriod	*	Exceptions for specified calendar dates, time-range based. Periods the station is operating/accessible. Additional to `regular_hours`. May overlap regular rules.
exceptional_closings	ExceptionalPeriod	*	Exceptions for specified calendar dates, time-range based. Periods the station is not operating/accessible. Overwriting `regular_hours` and `exceptional_openings`. Should not overlap `exceptional_openings`.

8.4.17.1. Example: 24/7 open with exceptional closing.

Open 24 hours per day, 7 days a week, except for 25th of December 2018 between 03:00 and 05:00.

```
{
  "twentyfourseven": true,
  "exceptional_closings": [{
    "period_begin": "2018-12-25T03:00:00Z",
    "period_end": "2018-12-25T05:00:00Z"
  }]
}
```

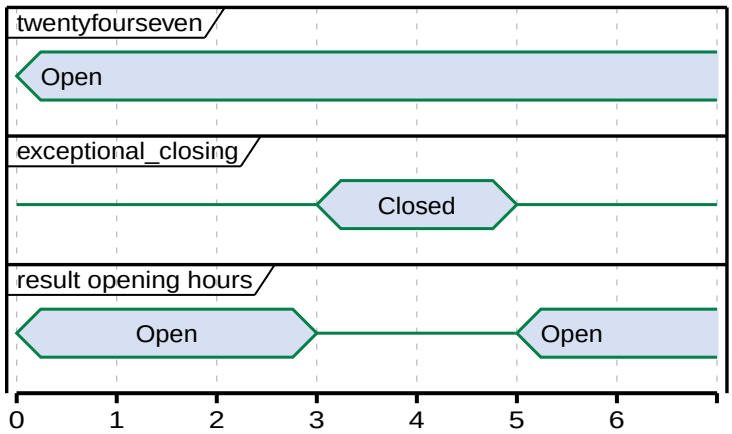


Figure 25. Diagram showing a representation of the example 24/7 open with exception closing.

8.4.17.2. Example: Opening Hours with exceptional closing.

Regular opening hours between 01:00 and 06:00. With exceptional closing on 25th of December 2018 between 03:00 and 05:00.

```
{
  "twentyfourseven": false,
  "regular_hours": [{
    "weekday": 1,
    "period_begin": "01:00",
    "period_end": "06:00"
  }, {
    "weekday": 2,
    "period_begin": "01:00",
    "period_end": "06:00"
  }],
  "exceptional_closings": [{
    "period_begin": "2018-12-25T03:00:00Z",
    "period_end": "2018-12-25T05:00:00Z"
  }]
}
```

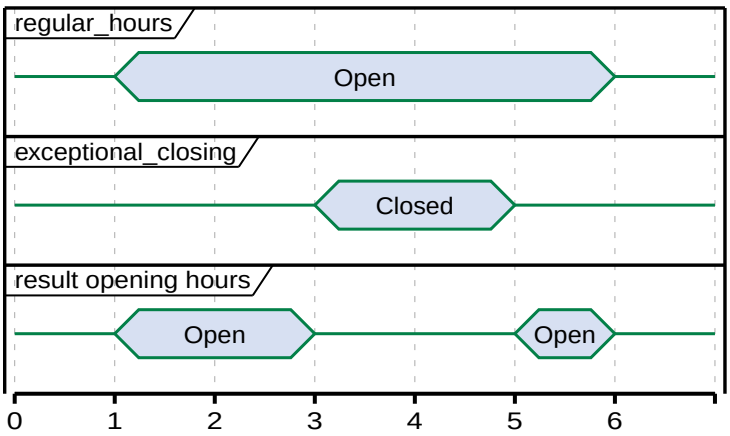


Figure 26. Diagram showing a representation of the example Opening Hours with exceptional closing

8.4.17.3. Example: Opening Hours with exceptional opening.

Regular opening hours between 00:00 and 04:00. With exceptional opening on 25th of December 2018 between 05:00 and 07:00.

```
{
  "twentyfourseven": false,
  "regular_hours": [{
    "weekday": 1,
    "period_begin": "00:00",
    "period_end": "04:00"
  }, {
    "weekday": 2,
    "period_begin": "00:00",
    "period_end": "04:00"
  }],
  "exceptional_openings": [{
    "period_begin": "2018-12-25T05:00:00Z",
    "period_end": "2018-12-25T06:00:00Z"
  }]
}
```

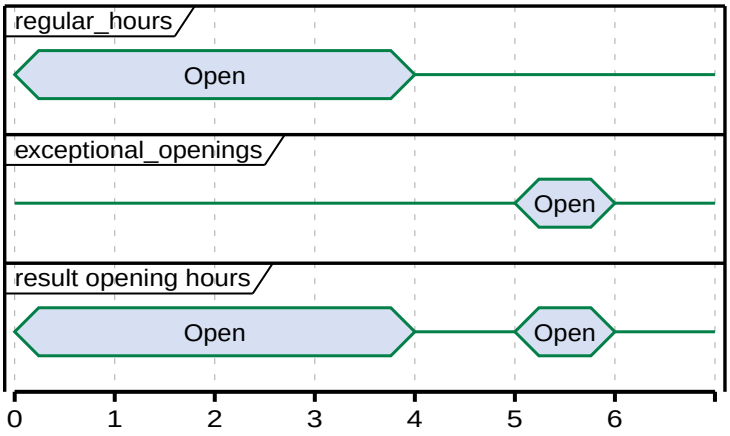


Figure 27. Diagram showing a representation of the example Opening Hours with exceptional opening.

8.4.18. Image *class*

This class references an image related to an EVSE in terms of a file name or url. According to the roaming connection between one EVSE Operator and one or more Navigation Service Providers, the hosting or file exchange of image payload data has to be defined. The exchange of this content data is out of scope of OCPI. However, the recommended setup is a public available web server hosted and updated by the EVSE Operator. Per charge point an unlimited number of images of each type is allowed. Recommended are at least two images where one is a network or provider logo and the second is a station photo. If two images of the same type are defined, not only one should be selected but both should be displayed together.

Photo Dimensions: The recommended dimensions for all photos is a minimum width of 800 pixels and a minimum height of 600 pixels. Thumbnail should always have the same orientation as the original photo with a size of 200 by 200 pixels.

Logo Dimensions: The recommended dimensions for logos are exactly 512 pixels in width height. Thumbnail representations of logos should be exactly 128 pixels in width and height. If not squared, thumbnails should have the same orientation as the original.

Property	Type	Card.	Description
url	URL	1	URL from where the image data can be fetched through a web browser.
thumbnail	URL	?	URL from where a thumbnail of the image can be fetched through a webbrowser.
category	ImageCategory	1	Describes what the image is used for.
type	CiString(4)	1	Image type like: gif, jpeg, png, svg.
width	int(5)	?	Width of the full scale image.
height	int(5)	?	Height of the full scale image.

8.4.19. ImageCategory *OpenEnum*

The category of an image to obtain the correct usage in a user presentation. The category has to be set accordingly to the image content in order to guarantee the right usage.

Value	Description
CHARGER	Photo of the physical device that contains one or more EVSEs.
ENTRANCE	Location entrance photo. Should show the car entrance to the location from street side.
LOCATION	Location overview photo.
NETWORK	Logo of an associated roaming network to be displayed with the EVSE for example in lists, maps and detailed information views.
OPERATOR	Logo of the charge point operator, for example a municipality, to be displayed in the EVSEs detailed information view or in lists and maps, if no network logo is present.
OTHER	Other
OWNER	Logo of the charge point owner, for example a local store, to be displayed in the EVSEs detailed information view.

8.4.20. ParkingDirection *enum*

Indicates the direction in which parking occurs relative to the roadway on which vehicles approach the EVSE.

Value	Description
PARALLEL	Parking happens parallel to the roadway on which vehicles approach the EVSE.
PERPENDICULAR	Parking happens perpendicular to the roadway on which vehicles approach the EVSE.
ANGLE	Parking happens at an angle to the roadway on which vehicles approach the EVSE (i.e. echelon parking).

8.4.21. ParkingRestriction *OpenEnum*

This value, if provided, represents the restriction to the parking spot for different purposes.

Value	Description
CUSTOMERS	Parking spot for customers or guests only, for example in case of a hotel or shop.
DISABLED	Reserved parking spot for disabled people with valid ID.
EMPLOYEES	Parking only for people who work at a site, building, or complex that the Location belongs to.
EV_ONLY	Reserved parking spot for electric vehicles.
MOTORCYCLES	Parking spot only suitable for (electric) motorcycles or scooters.
PLUGGED	Parking is only allowed while plugged in (charging).
TAXIS	Parking only for taxi vehicles.
TENANTS	Parking only for people who live in a complex that the Location belongs to.

8.4.22. ParkingType *OpenEnum*

Reflects the general type of the charge point's location. May be used for user information.

Value	Description
ALONG_MOTORWAY	Location on a parking facility/rest area along a motorway, freeway, interstate, highway etc.
PARKING_GARAGE	Multistorey car park.
PARKING_LOT	A cleared area that is intended for parking vehicles, i.e. at super markets, bars, etc.
ON_DRIVEWAY	Location is on the driveway of a house/building.
ON_STREET	Parking in public space along a street.
UNDERGROUND_GARAGE	Multistorey car park, mainly underground.

8.4.23. PowerType *enum*

Value	Description
AC_1_PHASE	AC single phase.
AC_2_PHASE	AC two phases, only two of the three available phases connected.
AC_2_PHASE_SPLIT	AC two phases using split phase system.
AC_3_PHASE	AC three phases.
DC	Direct Current.

8.4.24. PublishTokenType *class*

Defines the set of values that identify a token to which a Location might be published.

At least one of the following fields SHALL be set: `uid`, `visual_number`, or `group_id`.

When `uid` is set, `type` SHALL also be set.

When `visual_number` is set, `issuer` SHALL also be set.

Property	Type	Card.	Description
uid	CiString(36)	?	Unique ID by which this Token can be identified.
type	TokenType	?	Type of the token.
visual_number	string(64)	?	Visual readable number/identification as printed on the Token (RFID card).
issuer	string(64)	?	Issuing company, most of the times the name of the company printed on the token (RFID card), not necessarily the eMSP.

Property	Type	Card.	Description
group_id	CiString(36)	?	This ID groups a couple of tokens. This can be used to make two or more tokens work as one.

8.4.25. RegularHours *class*

Regular recurring operation or access hours.

Property	Type	Card.	Description
weekday	int(1)	1	Number of day in the week, from Monday (1) till Sunday (7)
period_begin	string(5)	1	Begin of the regular period, in local time, given in hours and minutes. Must be in 24h format with leading zeros. Example: "18:15". Hour/Minute separator: ":" Regex: `([0-1][0-9]|2[0-3]):[0-5][0-9]`.
period_end	string(5)	1	End of the regular period, in local time, syntax as for `period_begin`. Must be later than `period_begin`.

8.4.25.1. Example

Operating on weekdays from 8am till 8pm with one exceptional opening on 22/6/2014 and one exceptional closing the Monday after:

```
"opening_times": {
  "regular_hours": [
    {
      "weekday": 1,
      "period_begin": "08:00",
      "period_end": "20:00"
    },
    {
      "weekday": 2,
      "period_begin": "08:00",
      "period_end": "20:00"
    },
    {
      "weekday": 3,
      "period_begin": "08:00",
      "period_end": "20:00"
    },
    {
      "weekday": 4,
      "period_begin": "08:00",
      "period_end": "20:00"
    },
    {
      "weekday": 5,
      "period_begin": "08:00",
      "period_end": "20:00"
    }
  ],
  "twentyfourseven": false,
  "exceptional_openings": [
    {
      "period_begin": "2014-06-21T09:00:00Z",
      "period_end": "2014-06-21T12:00:00Z"
```

```
      }
    ],
    "exceptional_closings": [
      {
        "period_begin": "2014-06-24T00:00:00Z",
        "period_end": "2014-06-25T00:00:00Z"
      }
    ]
  }
```

This represents the following schedule, where ~~stroked out~~ days are without operation hours, **bold** days are where exceptions apply and regular displayed days are where the regular schedule applies.

Week day	**Mo**	**Tu**	**We**	**Th**	**Fr**	**Sa**	**Su**	**Mo**	**Tu**	**We**	**Th**	**Fr**	**Sa**	**Su**
Date	16	17	18	19	20	**21**	~~22~~	23	~~**24**~~	25	26	27	~~28~~	~~29~~
Open from	08	08	08	08	08	09	-	08	-	08	08	08	-	-
Open till	20	20	20	20	20	12	-	20	-	20	20	20	-	-

8.4.26. Status *enum*

The status of an EVSE.

Value	**Description**
AVAILABLE	The EVSE/Connector is able to start a new charging session.
BLOCKED	The EVSE/Connector is not accessible because of a physical barrier, i.e. a car.
CHARGING	The EVSE/Connector is in use.
INOPERATIVE	The EVSE/Connector is not yet active, or temporarily not available for use, but not broken or defect.
OUTOFORDER	The EVSE/Connector is currently out of order, some part/components may be broken/defect.
PLANNED	The EVSE/Connector is planned, will be operating soon.
REMOVED	The EVSE/Connector was discontinued/removed.
RESERVED	The EVSE/Connector is reserved for a particular EV driver and is unavailable for other drivers.
UNKNOWN	No status information available (also used when offline).

8.4.27. StatusSchedule *class*

This type is used to schedule status periods in the future. The eMSP can provide this information to the EV user for trip planning purposes. A period MAY have no end. Example: "This station will be running as of tomorrow. Today it is still planned and under construction."

Property	Type	Card.	Description
period_begin	DateTime	1	Begin of the scheduled period.
period_end	DateTime	?	End of the scheduled period, if known.
status	Status	1	Status value during the scheduled period.

NOTE The scheduled status is purely informational. When the status actually changes, the CPO must push an update to the EVSEs `status` field itself.

8.4.28. VehicleType *OpenEnum*

A categorization of vehicles to indicate which type of vehicles can use a certain EVSE. Approximate UNECE codes corresponding to our categories are given in the third column in the table.

Value	Description	UNECE Code
MOTORCYCLE	A motorcycle	L
PERSONAL_VEHICLE	A personal vehicle, a passenger car	M1
PERSONAL_VEHICLE_WITH_TRAILER	A personal vehicle with a trailer attached	M1 + O
VAN	A light-duty van with a height smaller than 275 cm	N1
SEMI_TRACTOR	A heavy-duty tractor unit without a trailer	T
RIGID	A heavy-duty truck without an articulation point	N2 (under 12 tonnes) / N3 (over 12 tonnes)
TRUCK_WITH_TRAILER	A heavy-duty truck (tractor or rigid) with a trailer attached	N2/N3 + O
BUS	A bus or a motor coach.	M2 (under 5 tonnes) / M3 (over 5 tonnes)
DISABLED	A vehicle with a permit for parking spaces for people with disabilities	M1 (assuming that these are typically based on personal vehicles)

NOTE It may seem surprising that OCPI uses a custom vehicle categorization scheme rather than one defined in another specification. During OCPI 3.0 development it appeared however that existing classifications, like the UNECE Classification and Definition of Vehicles, are overly detailed and technical and offer little help in making clear which vehicles can use a certain EVSE. For OCPI 3.0 we opted for a deliberately common sense based categorization that we believe will be easier to use for Drivers and CPOs.

9. *Sessions* module

Module Identifier: `sessions`

Data owner: `CPO`

Type: Functional Module

The Session object describes one charging session. The Session object is owned by the CPO back-end system, and can be GET from the CPO system, or pushed by the CPO to another system.

9.1. Flow and Lifecycle

9.1.1. Push model

When the CPO creates a Session object they push it to the corresponding eMSP by calling PUT on the eMSP's Sessions endpoint with the newly created Session object.

Any changes to a Session in the CPO system are sent to the eMSP system by calling PATCH on the eMSP's Sessions endpoint with the updated Session object.

Sessions cannot be deleted, final status of a session is: `COMPLETED`.

When the CPO is not sure about the state or existence of a Session object in the eMSP's system, the CPO can call GET on the eMSP's Sessions endpoint to validate the Session object in the eMSP's system.

9.1.2. Pull model

eMSPs who do not support the Push model need to call GET on the CPO's Sessions endpoint to receive a list of Sessions.

This GET method can also be used in combination with the Push model to retrieve Sessions after the system (re-)connects to a CPO, to get a list Sessions *missed* during a downtime of the eMSP's system.

9.1.3. Set: Charging Preferences

For a lot of smart charging use cases, input from the driver is needed. The smart charging algorithms need to be able to give certain session priority over others. In other words they need to know how much energy an EV needs before what time. Via a PUT request on the Sender Interface, during an ongoing session, the eMSP can send Charging Preferences on behalf of the driver.

The eMSP can determine if an EVSE supports Charging Preferences by checking if the EVSE capabilities contains: CHARGING_PREFERENCES_CAPABLE.

Via Tariffs the CPO can give different Charging Preferences different prices. A Connector can have multiple Tariffs, one for each ProfileType.

9.1.4. Reservation

When a EV driver makes a Reservation for a Charge Point/EVSE, the Sender SHALL create a new Session object with `status` = `RESERVED` When the Push model is used, the CPO SHALL push the new Session object to the Receiver.

When a reservation results in a charging session for the same `Token`, the Session object `status` to: `ACTIVE`

When a reservation does not result in a charging session, the Session object `status` SHALL be set to: `COMPLETED`.

A CDR might be created even if no energy was transferred to the EV, just for the costs of the reservation.

9.2. Interfaces and Endpoints

9.2.1. Sender Interface

Typically implemented by market roles like: CPO.

Method	Description
GET	Fetch Session objects of charging sessions last updated between the `{date_from}` and `{date_to}` (paginated).
POST	n/a
PUT	Setting Charging Preferences of an ongoing session.
PATCH	n/a
DELETE	n/a

9.2.1.1. GET Method

Fetch Sessions from a CPO system.

Endpoint structure definition:

`{sessions_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/cpo/2.2.1/sessions/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/sessions/?offset=50`

`https://www.server.com/ocpi/2.2.1/sessions/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/cpo/2.2.1/sessions/?offset=50&limit=100`

Request Parameters

Only Sessions with `last_update` between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned.

This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	yes	Only return Sessions that have `last_updated` after or equal to this Date/Time (inclusive).

Parameter	Datatype	Required	Description
date_to	DateTime	no	Only return Sessions that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

Response Data

The response contains a list of Session objects that match the given parameters in the request, the header will contain the pagination related headers.

Any older information that is not specified in the response is considered no longer valid. Each object must contain all required fields. Fields that are not specified may be considered as null values.

Datatype	Card.	Description
Session	*	List of Session objects that match the request parameters.

9.2.1.2. PUT Method

Set/update the driver's Charging Preferences for this charging session.

Endpoint structure definition:

`{sessions_endpoint_url}/{session_id}/charging_preferences`

Examples:

`https://www.server.com/ocpi/cpo/2.2.1/sessions/1234/charging_preferences`

NOTE The `/charging_preferences` URL suffix is required when setting Charging Preferences.

Request Parameters

The following parameter has to be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	Session.id of the Session for which the Charging Preferences are to be set.

Request Body

In the body, a ChargingPreferences object has to be provided.

Type	Card.	Description
ChargingPreferences	1	Updated Charging Preferences of the driver for this Session.

Response Data

The response contains a ChargingPreferencesResponse value.

Type	Card.	Description
ChargingPreferencesResponse	1	Response to the Charging Preferences PUT request.

9.2.2. Receiver Interface

Typically implemented by market roles like: eMSP and SCSP.

Sessions are Client Owned Objects, so the endpoints need to contain the required extra fields: {party_id} and {country_code}.

Endpoint structure definition:

`{sessions_endpoint_url}/{country_code}/{party_id}/{session_id}`

Example:

`https://www.server.com/ocpi/emsp/2.2.1/sessions/BE/BEC/1234`

Method	Description
GET	Retrieve a Session object from the eMSP's system with Session.id equal to `{session_id}`.
POST	n/a
PUT	Send a new/updated Session object to the eMSP.
PATCH	Update the Session object with Session.id equal to `{session_id}`.
DELETE	n/a

9.2.2.1. GET Method

The CPO system might request the current version of a Session object from the eMSP's system to, for example, validate the state, or because the CPO has received an error during a PATCH operation.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO performing the GET on the eMSP's system.

Parameter	Datatype	Required	Description
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO performing the GET on the eMSP's system.
session_id	CiString(36)	yes	id of the Session object to get from the eMSP's system.

Response Data

The response contains the requested Session object.

Datatype	Card.	Description
Session	1	Requested Session object.

9.2.2.2. PUT Method

Inform the eMSP's system about a new/updated Session object in the CPO's system.

When a PUT request is received for an existing Session object (the object is PUT to the same URL), The newly received Session object SHALL replace the existing object.

Any `charging_periods` from the existing object SHALL be replaced by the `charging_periods` from the newly received Session object. If the new Session object does not contain `charging_periods` (field is omitted or contains any empty list), the `charging_periods` of the existing object SHALL be removed (replaced by the new empty list).

Request Body

The request contains the new or updated Session object.

Type	Card.	Description
Session	1	New or updated Session object.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO performing this PUT on the eMSP's system. This SHALL be the same value as the `country_code` in the Session object being pushed.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO performing this PUT on the eMSP's system. This SHALL be the same value as the `party_id` in the Session object being pushed.
session_id	CiString(36)	yes	id of the new or updated Session object.

9.2.2.3. PATCH Method

Same as the PUT method, but only the fields/objects that need to be updated have to be present. Fields/objects which are not specified are considered unchanged.

Any request to the PATCH method SHALL contain the `last_updated` field.

The PATCH method of the Session Receiver interface works on the entire Session object only. It is not allowed to use extra URL segments to try to PATCH fields of inner objects of the Session object directly.

When a PATCH request contains the `charging_periods` field (inside a Session object), this SHALL be processed as a request to add all the ChargingPeriod objects to the existing Session object. If the request `charging_periods` list is omitted (or contains an empty list), no changes SHALL be made to the existing list of `charging_periods`.

If existing ChargingPeriod objects in a Session need to be replaced or removed, the Sender SHALL use the PUT method to replace the entire Session object (including all the `charging_periods`).

Example: update the total cost

Patching the `total_cost` needs to be done on the Session Object.

```
PATCH https://www.server.com/ocpi/cpo/2.2.1/sessions/NL/TNM/101

{
  "total_cost": {
    "before_taxes": 0.60,
    "taxes": [{
      "name": "VAT",
      "amount": 0.06
    }]
  },
  "last_updated": "2019-06-23T08:11:00Z"
}
```

Example: adding a new ChargingPeriod

PATCH used to add a new ChargingPeriod to the Session and updating all related fields.

```
PATCH https://www.server.com/ocpi/cpo/2.2.1/sessions/NL/TNM/101

{
  "kwh": 15.00,
  "charging_periods": [{
    "start_date_time": "2019-06-23T08:16:02Z",
    "dimensions": [{
      "type": "ENERGY",
      "volume": 2200
    }]
  }],
  "total_cost": {
    "before_taxes": 0.80,
    "taxes": [{
      "name": "VAT",
      "amount": 0.08
    }]
  },
  "last_updated": "2019-06-23T08:16:02Z"
}
```

9.3. Object description

9.3.1. *Session* Object

The Session object describes one charging session. That doesn't mean it is required that energy has been transferred between EV and the Charge Point. It is possible that the EV never took energy from the Charge Point because it was instructed not to take energy by the driver. But as the EV was connected to the Charge Point, some form of start tariff, park tariff or reservation cost might be relevant.

NOTE Although OCPI supports such pricing mechanisms, local laws might not allow this.

It is recommended to add enough `ChargingPeriods` to a Session so that the eMSP is able to provide feedback to the EV driver about the progress of the charging session. The ideal amount of transmitted Charging Periods depends on the charging speed. The Charging Periods should be sufficient for useful feedback but they should not generate too much unneeded traffic either. How many Charging Periods are transmitted is left to the CPO to decide. The following are just some points to consider:

- Adding a new Charging Period every minute for an AC charging session can be too much as it will yield 180 Charging Periods for an (assumed to be) average 3h session.
- A new Charging Period every 30 minutes for a DC fast charging session is not enough as it will yield only one Charging Period for an (assumed to be) average 30min session.

It is also recommended to add Charging Periods for all moments that are relevant for the Tariff changes, see CDR object description for more information.

For more information about how `step_size` impacts the calculation of the cost of charging also see the CDR object description.

Property	Type	Card.	Description
country_code	CiString(2)	1	ISO-3166 alpha-2 country code of the CPO that 'owns' this Session.
party_id	CiString(3)	1	ID of the CPO that 'owns' this Session (following the ISO-15118 standard).
id	CiString(36)	1	The unique id that identifies the charging session in the CPO platform.
start_date_time	DateTime	1	The timestamp when the session became ACTIVE in the Charge Point. When the session is still PENDING, this field SHALL be set to the time the Session was created at the Charge Point. When a Session goes from PENDING to ACTIVE, this field SHALL be updated to the moment the Session went to ACTIVE in the Charge Point.
end_date_time	DateTime	?	The timestamp when the session was completed/finished, charging might have finished before the session ends, for example: EV is full, but parking cost also has to be paid.

Property	Type	Card.	Description
kwh	number	1	How many kWh were charged.
cdr_token	CdrToken	1	Token used to start this charging session, including all the relevant information to identify the unique token.
auth_method	AuthMethod	1	Method used for authentication. This might change during a session, for example when the session was started with a reservation: ReserveNow: `COMMAND`. When the driver arrives and starts charging using a Token that is whitelisted: `WHITELIST`.
authorization_reference	CiString(36)	?	Reference to the authorization given by the eMSP. When the eMSP provided an `authorization_reference` in either: real-time authorization, StartSession or ReserveNow this field SHALL contain the same value. When different `authorization_reference` values have been given by the eMSP that are relevant to this Session, the last given value SHALL be used here.
location_id	CiString(36)	1	Location.id of the Location object of this CPO, on which the charging session is/was happening.
evse_uid	CiString(36)	1	EVSE.uid of the EVSE of this Location on which the charging session is/was happening. Allowed to be set to: `#NA` when this session is created for a reservation, but no EVSE yet assigned to the driver.
connector_id	CiString(36)	1	Connector.id of the Connector of this Location where the charging session is/was happening. Allowed to be set to: `#NA` when this session is created for a reservation, but no connector yet assigned to the driver.
meter_id	string(255)	?	Optional identification of the kWh meter.
currency	string(3)	1	ISO 4217 code of the currency used for this session.
charging_periods	ChargingPeriod	*	An optional list of Charging Periods that can be used to calculate and verify the total cost.
total_cost	Price	?	The total cost of the session in the specified currency. This is the price that the eMSP will have to pay to the CPO. A total_cost of 0.00 means free of charge. When omitted, i.e. no price information is given in the Session object, it does not imply the session is/was free of charge.
status	SessionStatus	1	The status of the session.
last_updated	DateTime	1	Timestamp when this Session was last updated (or created).

NOTE Different `authorization_reference` values might happen when for example a ReserveNow had a different `authorization_reference` then the value returned by a real-time authorization.

9.3.1.1. Examples

Simple Session example of just starting a session

```
{
    "country_code": "NL",
    "party_id": "STK",
    "id": "101",
    "start_date_time": "2020-03-09T10:17:09Z",
    "kwh": 0.0,
    "cdr_token": {
        "country_code": "NL",
        "party_id": "TST",
        "uid": "123abc",
        "type": "RFID",
        "contract_id": "NL-TST-C12345678-S"
    },
    "auth_method": "WHITELIST",
    "location_id": "LOC1",
    "evse_uid": "3256",
    "connector_id": "1",
    "currency": "EUR",
    "total_cost": {
        "before_taxes": 2.5
    },
    "status": "PENDING",
    "last_updated": "2020-03-09T10:17:09Z"
}
```

Simple Session example of a short finished session

```
{
  "country_code": "BE",
  "party_id": "BEC",
  "id": "101",
  "start_date_time": "2015-06-29T22:39:09Z",
  "end_date_time": "2015-06-29T23:50:16Z",
  "kwh": 41.12,
   "cdr_token": {
        "country_code": "NL",
        "party_id": "TST",
        "uid": "123abc",
        "type": "RFID",
        "contract_id": "NL-TST-C12345678-S"
    },
  "auth_method": "WHITELIST",
  "location_id": "LOC1",
  "evse_uid": "3256",
  "connector_id": "1",
  "currency": "EUR",
  "charging_periods": [{
    "start_date_time": "2015-06-29T22:39:09Z",
    "dimensions": [{
      "type": "ENERGY",
      "volume": 120
    }, {
      "type": "MAX_CURRENT",
      "volume": 30
    }]
  }, {
    "start_date_time": "2015-06-29T22:40:54Z",
    "dimensions": [{
      "type": "ENERGY",
      "volume": 41000
    }, {
      "type": "MIN_CURRENT",
```

```
      "volume": 34
    }]
  }, {
    "start_date_time": "2015-06-29T23:07:09Z",
    "dimensions": [{
      "type": "PARKING_TIME",
      "volume": 0.718
    }],
    "tariff_id": "12"
  }],
  "total_cost": {
    "before_taxes": 8.50,
    "taxes": [{
      "name": "VAT",
      "amount": 0.85
    }]
  },
  "status": "COMPLETED",
  "last_updated": "2015-06-29T23:50:17Z"
}
```


9.3.2. *ChargingPreferences* Object

Contains the charging preferences of an EV driver.

Property	Type	Card.	Description
profile_type	ProfileType	1	Type of Smart Charging Profile selected by the driver. The ProfileType has to be supported at the Connector and for every supported ProfileType, a Tariff MUST be provided. This gives the EV driver the option between different pricing options.
departure_time	DateTime	?	Expected departure. The driver has given this Date/Time as expected departure moment. It is only an estimation and not necessarily the Date/Time of the actual departure.
energy_need	number	?	Requested amount of energy in kWh. The EV driver wants to have this amount of energy charged.
discharge_allowed	boolean	?	The driver allows their EV to be discharged when needed, as long as the other preferences are met: EV is charged with the preferred energy (`energy_need`) until the preferred departure moment (`departure_time`). Default if omitted: **false**

9.4. Data types

9.4.1. ChargingPreferencesResponse *enum*

An enum with possible responses to a PUT Charging Preferences request.

If a PUT with `ChargingPreferences` is received for an EVSE that does not have the capability `CHARGING_PREFERENCES_CAPABLE`, the receiver should respond with an HTTP status of 404 and an OCPI status code of 2001 in the OCPI response object.

Value	Description
ACCEPTED	Charging Preferences accepted, EVSE will try to accomplish them, although this is no guarantee that they will be fulfilled.
DEPARTURE_REQUIRED	CPO requires `departure_time` to be able to perform Charging Preference based Smart Charging.
ENERGY_NEED_REQUIRED	CPO requires `energy_need` to be able to perform Charging Preference based Smart Charging.
NOT_POSSIBLE	Charging Preferences contain a demand that the EVSE knows it cannot fulfill.
PROFILE_TYPE_NOT_SUPPORTED	`profile_type` contains a value that is not supported by the EVSE.

9.4.2. ProfileType *enum*

Different smart charging profile types.

Value	Description
CHEAP	Driver wants to use the cheapest charging profile possible.
FAST	Driver wants his EV charged as quickly as possible and is willing to pay a premium for this, if needed.
GREEN	Driver wants his EV charged with as much regenerative (green) energy as possible.
REGULAR	Driver does not have special preferences.

9.4.3. SessionStatus *enum*

Defines the state of a session.

Value	Description
ACTIVE	The session has been accepted and is active. All pre-conditions were met: Communication between EV and EVSE (for example: cable plugged in correctly), EV or driver is authorized. EV is being charged, or can be charged. Energy is, or is not, being transfered.
COMPLETED	The session has been finished successfully. No more modifications will be made to the Session object using this state.
INVALID	The Session object using this state is declared invalid and will not be billed.
PENDING	The session is pending, it has not yet started. Not all pre-conditions are met. This is the initial state. The session might never become an *active* session.
RESERVATION	The session is started due to a reservation, charging has not yet started. The session might never become an *active* session.

10. *CDRs* module

Module Identifier: `cdrs`

Data owner: `CPO`

Type: Functional Module

A **Charge Detail Record** is the description of a concluded charging session. The CDR is the only billing-relevant object. CDRs are sent from the CPO to the eMSP after the charging session has ended. Although there is no requirement to send CDRs in (semi-) realtime, it is seen as good practice to send them as soon as possible. But if there is an agreement between parties to send them, for example, once a month, that is also allowed by OCPI.

10.1. Flow and Lifecycle

CDRs are created by the CPO. They most likely will be sent only to the eMSP that needs to pay the bill of the underlying charging session. Because a CDR is for billing purposes, it cannot be changed or replaced once sent to the eMSP. Changes are simply not allowed. Instead, a Credit CDR can be sent.

CDRs may be sent for charging locations that have not been published via the Location module. This is typically for home chargers.

10.1.1. Credit CDRs

As CDRs are used for billing and can be seen as a kind of invoice, they cannot be deleted. Instead, they have to be credited.

When a CPO wants to make changes to a CDR that was already sent to the eMSP, the CPO has to send a Credit CDR for the first CDR. This credit CDR SHALL have a different CDR.id which can be a completely different number, or it can be the id of the original CDR with something appended like for example: `-C` to make it unique again. To indicate that a CDR is a Credit CDR, the `credit` field has to be set to `true`. The Credit CDR references the old CDR via the `credit_reference_id` field, which SHALL contain the `id` of the original CDR. The Credit CDR will contain all the data of the original CDR. Only the values in the `total_cost` field SHALL contain the negative amounts of the original CDR.

After having sent the Credit CDR, the CPO can send a new CDR with a new unique ID and the fields: `credit` and `credit_reference_id` omitted.

NOTE	How far back in time a CPO can send a Credit CDR is not defined by OCPI. It is up the business contracts between the different parties involved, as there might be local laws involved etc.

10.1.2. Push model

When the CPO creates CDR(s) they push them to the relevant eMSP by calling POST on the eMSPs CDRs endpoint with the newly created CDR(s). A CPO is not required to send *all* CDRs to *all* eMSPs, it is allowed to only send CDRs to the eMSP that a CDR is relevant to.

CDRs should contain enough information (dimensions) to allow the eMSP to validate the total cost. It is advised to send enough information to the eMSP so that they can calculate their own costs for billing their customers. An eMSP might have a very different contract/pricing model with their EV drivers than the tariff structure of the CPO.

If the CPO, for any reason, wants to view a CDR it has posted to an eMSP's system, the CPO can retrieve the CDR by performing a GET request on the eMSP's CDRs endpoint at the URL returned in the response to the POST.

10.1.3. Pull model

eMSPs who do not support the Push model need to call GET on the CPO's CDRs endpoint to receive a list of CDRs.

This GET can also be used in combination with the Push model to retrieve CDRs after the system (re-)connects to a CPO, to get a list of CDRs *missed* during a downtime of the eMSP's system.

A CPO is not required to return all known CDRs, the CPO is allowed to return only the CDRs that are relevant for the requesting eMSP.

10.2. Interfaces and Endpoints

There are both, a Sender and a Receiver interface for CDRs. Depending on business requirements, parties can decide to use the Sender Interface (Pull model), or the Receiver Interface (Push model), or both. Push is the preferred model to use, because the Receiver will receive CDRs in semi-realtime when they are created by the CPO.

10.2.1. Sender Interface

Typically implemented by market roles like: CPO.

The CDRs endpoint can be used to retrieve CDRs.

Endpoint structure definition:

`{cdr_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/cpo/2.2.1/cdrs/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/cdrs/?offset=50`

`https://www.server.com/ocpi/2.2.1/cdrs/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/cpo/2.2.1/cdrs/?offset=50&limit=100`

Method	Description
GET	Fetch CDRs last updated (which in the current version of OCPI can only be the creation Date/Time) between the `{date_from}` and `{date_to}` (paginated).
POST	n/a
PUT	n/a
PATCH	n/a
DELETE	n/a

10.2.1.1. GET Method

Fetch CDRs from the CPO's system.

Request Parameters

If additional parameters: `{date_from}` and/or `{date_to}` are provided, only CDRs with `last_updated` between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned.

This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	no	Only return CDRs that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	no	Only return CDRs that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

Response Data

The endpoint returns a list of CDRs matching the given parameters in the GET request, the header will contain the pagination related headers.

Any older information that is not specified in the response is considered no longer valid. Each object must contain all required fields. Fields that are not specified may be considered as null values.

Datatype	Card.	Description
CDR	*	List of CDRs.

10.2.2. Receiver Interface

Typically implemented by market roles like: eMSP.

The CDRs endpoint can be used to create and retrieve CDRs.

Method	Description
GET	Retrieve an existing CDR.
POST	Send a new CDR.
PUT	n/a (CDRs cannot be replaced)
PATCH	n/a (CDRs cannot be updated)
DELETE	n/a (CDRs cannot be removed)

10.2.2.1. GET Method

Fetch CDRs from the receivers system.

Endpoint structure definition:

No structure defined. This is open to the eMSP to define, the URL is provided to the CPO by the eMSP in the result of the POST request. Therefore, OCPI does not define variables.

Example:

`https://www.server.com/ocpi/2.2.1/cdrs/1234`

Response URL

To retrieve an existing URL from the eMSP's system, the URL, returned in the response to a POST of a new CDR, has to be used.

Response Data

The endpoint returns the requested CDR, if it exists.

Datatype	Card.	Description
CDR	1	Requested CDR object.

10.2.2.2. POST Method

Creates a new CDR.

The POST method should contain the full and final CDR object.

Endpoint structure definition:

`{cdr_endpoint_url}`

Example:

`https://www.server.com/ocpi/2.2.1/cdrs/`

Request Body

In the POST request the new CDR object is sent.

Type	Card.	Description
CDR	1	New CDR object.

Response Headers

The response should contain the URL to the just created CDR object in the eMSP's system.

HTTP Header	Datatype	Required	Description
Location	URL	yes	URL to the newly created CDR in the eMSP's system, can be used by the CPO system to perform a GET on the same CDR.

The eMSP returns the URL where the newly created CDR can be found. OCPI does not define a specific structure for

this URL.

Example:

`https://www.server.com/ocpi/emsp/2.2.1/cdrs/123456`

10.3. Object description

10.3.1. *CDR* Object

The *CDR* object describes the charging session and its costs, how these costs are composed, etc.

The CDR object is different from the Session object. The Session object is dynamic as it reflects the current state of the charging session. The information is meant to be viewed by the driver while the charging session is ongoing.

The CDR on the other hand can be thought of as *sealed*, preserving the information valid at the moment in time the underlying session was started. This is a requirement of the main use case for CDRs, namely invoicing. If e.g. a street is renamed the day after a session took place, the driver should be presented with the name valid at the time the session was started. This guarantees that the CDR will be recognized as correct by the driver and is not going to be contested.

The *CDR* object shall always contain information like Location, EVSE, Tariffs and Token as they were **at the start** of the charging session.

ChargingPeriod: A CPO SHALL at least start (and add) a ChargingPeriod every moment/event that has relevance for the total costs of a CDR. During a charging session, different parameters change all the time, like the amount of energy used, or the time of day. These changes can result in another Price Component of the Tariff becoming active. When another Price Component becomes active, the CPO SHALL add a new Charging Period with at least all the relevant information for the change to the other Price Component. The CPO is allowed to add more *in-between* Charging Periods to a CDR though.

Examples of additional Charging Periods that are required to be added because another Price Component is becoming active:

- When an energy changes in price after 17:00. The CPO has to start a new Charging Period at 17:00. The CPO also has to list the energy in kWh consumed until 17:00 in the Charging Period that ends at 17:00.
- When the price of a energy is higher when the EV is charging faster than 32A, a new Charging Period has to be added the moment the charging power goes over 32A. This may be a moment that is calculated by the CPO, as the Charge Point might not send the information to the CPO, but it can be interpolated by the CPO using the metering information before and after that moment.

step_size: When calculating the cost of a charging session, `step_size` SHALL only be taken into account once per session for the TariffDimensionType `ENERGY` and once for `PARKING_TIME` and `TIME` combined.

`step_size` is not taken into account when switching time based paying for charging to paying for parking (charging has stopped but EV still connected).
Example: `step_size` for both charging (`TIME`) and parking is 5 minutes. After 21 minutes of charging, the EV is full but remains connected for 7 more minutes. The cost of charging will be calculated based on 21 minutes (not 25). The cost of parking will be calculated based on 10 minutes (`step_size` is 5).

`step_size` is not taken into account when switching from (for example) one `ENERGY` based tariff element to another. This is also true when switch from one (`TIME`) based tariff element to another (`TIME`) based tariff element, and one

PARKING_TIME tariff element to another PARKING_TIME based tariff element.
Example: when charging is more expensive after 17:00. The step_size of the tariff before 17:00 will not be used when charging starts before 17:00 and ends after 17:00. Only the step_size of the tariff (PriceComponent) after 17:00 is taken into account, for the total of the same amount for the session.

The step_size for the PriceComponent that is used to calculate the cost of such a 'last' ChargingPeriod SHALL be used. If the step_size differs for the different TariffElements, the step_size of the last relevant PriceComponent is used.

The step_size is not taken into account when switching between two Tariffs
Example: A driver selects a different Charging Preference profile_type during an ongoing charging session, the different profile might have a different tariff.

The step_size uses the total amount of a certain unit used during a session, not only the last ChargingPeriod. In other words, when the price of energy per kWh or the price of time per hour differs during a session,only the total amount of energy or time is used in calculations with step_size.
Example: Energy costs € 0.20 perkWh before 17:00 and € 0.27 per kWh after 17:00. Both Price Components have a step_size of 500 Wh. If a driver charges 4.3 kWh before 17:00 and 1.1 kWh after 17:00, a total of 5.4 kWh is charged. The step_size rounds this up to 5.5 kWh total. It does NOT round the energy used after 17:00 to 1.5 kWh.

Example: Time costs € 5 per hour before 17:00 and € 7 per hour after 17:00. Both Price Components have a step_size of 10 minutes. If a driver charges 6 minutes before 17:00 and 22 minutes after 17:00, this makes a total of 28 minutes charging. The step_size rounds this up to 30 minutes total, so 24 minutes after 17:00 will be billed. It does NOT round the minutes after 17:00 to 30 minutes, which would have made a total of 36 minutes.

In the cases that TIME and PARKING_TIME Tariff Elements are both used, step_size is only taken into account for the total parking duration `
Example: Time spent charging costs € 1.00 per hour and time spent parking (not charging) costs € 2.00 per hour. Both Price Components have a step_size of 10 minutes. If a driver charges 21 minutes, and keeps his EV connected while it is full for another 16 minutes, then the step_size rounds the parking duration up to 20 minutes, making it a total of 41 minutes. Note that the charging duration is not rounded up, as it is followed by another time based period.

Property	Type	Card.	Description
country_code	CiString(2)	1	ISO-3166 alpha-2 country code of the CPO that 'owns' this CDR.
party_id	CiString(3)	1	ID of the CPO that 'owns' this CDR (following the ISO-15118 standard).
id	CiString(39)	1	Uniquely identifies the CDR, the ID SHALL be unique per country_code/party_id combination. This field is longer than the usual 36 characters to allow for credit CDRs to have something appended to the original ID. Normal (non-credit) CDRs SHALL only have an ID with a maximum length of 36.
start_date_time	DateTime	1	Start timestamp of the charging session, or in-case of a reservation (before the start of a session) the start of the reservation.

Property	Type	Card.	Description
end_date_time	DateTime	1	The timestamp when the session was completed/finished, charging might have finished before the session ends, for example: EV is full, but parking cost also has to be paid.
session_id	CiString(36)	?	Unique ID of the Session for which this CDR is sent. Is only allowed to be omitted when the CPO has not implemented the Sessions module or this CDR is the result of a reservation that never became a charging session, thus no OCPI Session.
cdr_token	CdrToken	1	Token used to start this charging session, including all the relevant information to identify the unique token.
auth_method	AuthMethod	1	Method used for authentication. Multiple <mod_cdrs_authmethod_enum,AuthMethods>> are possible during a charging sessions, for example when the session was started with a reservation: ReserveNow: `COMMAND`. When the driver arrives and starts charging using a Token that is whitelisted: `WHITELIST`. The last method SHALL be used in the CDR.
authorization_reference	CiString(36)	?	Reference to the authorization given by the eMSP. When the eMSP provided an `authorization_reference` in either: real-time authorization, StartSession or ReserveNow, this field SHALL contain the same value. When different `authorization_reference` values have been given by the eMSP that are relevant to this Session, the last given value SHALL be used here.
cdr_location	CdrLocation	1	Location where the charging session took place, including only the relevant EVSE and Connector.
meter_id	string(255)	?	Identification of the Meter inside the Charge Point.
currency	string(3)	1	Currency of the CDR in ISO 4217 Code.
tariffs	Tariff	*	List of relevant Tariffs, see: Tariff. When relevant, a *Free of Charge* tariff should also be in this list, and point to a defined *Free of Charge* Tariff.
charging_periods	ChargingPeriod	+	List of Charging Periods that make up this charging session.
signed_data	SignedData	?	Signed data that belongs to this charging Session.
total_cost	Price	1	Total sum of all the costs of this transaction in the specified currency.
total_fixed_cost	Price	?	Total sum of all the fixed costs in the specified currency, except fixed price components of parking and reservation. The cost not depending on amount of time/energy used etc. Can contain costs like a start tariff.
total_energy	number	1	Total energy charged, in kWh.

Property	Type	Card.	Description
total_energy_cost	Price	?	Total sum of all the cost of all the energy used, in the specified currency.
total_time	number	1	Total duration of the charging session (including the duration of charging and not charging), in hours.
total_time_cost	Price	?	Total sum of all the cost related to duration of charging during this transaction, in the specified currency.
total_parking_time	number	?	Total duration of the charging session where the EV was not charging (no energy was transferred between EVSE and EV), in hours.
total_parking_cost	Price	?	Total sum of all the cost related to parking of this transaction, including fixed price components, in the specified currency.
total_reservation_cost	Price	?	Total sum of all the cost related to a reservation of a Charge Point, including fixed price components, in the specified currency.
remark	string(255)	?	Optional remark, can be used to provide additional human readable information to the CDR, for example: reason why a transaction was stopped.
invoice_reference_id	CiString(39)	?	This field can be used to reference an invoice, that will later be send for this CDR. Making it easier to link a CDR to a given invoice. Maybe even group CDRs that will be on the same invoice.
credit	boolean	?	When set to `true`, this is a Credit CDR, and the field `credit_reference_id` needs to be set as well.
credit_reference_id	CiString(39)	?	Is required to be set for a Credit CDR. This SHALL contain the `id` of the CDR for which this is a Credit CDR.
home_charging_compensation	boolean	?	When set to `true`, this CDR is for a charging session using the home charger of the EV Driver for which the energy cost needs to be financial compensated to the EV Driver.
last_updated	DateTime	1	Timestamp when this CDR was last updated (or created).

NOTE The actual charging duration (energy being transferred between EVSE and EV) of a charging session can be calculated: `total_charging_time = total_time - total_parking_time`.

NOTE Having both a `credit` and a `credit_reference_id` might seem redundant. But it is seen as an advantage as a boolean flag used in queries is much faster than simple string comparison of references.

NOTE Different `authorization_reference` values might happen when for example a ReserveNow had a different `authorization_reference` then the value returned by a real-time authorization.

NOTE

When no `start_date_time` and/or `end_date_time` is known to the CPO, normally the CPO cannot send the CDR. If the MSP and CPO both agree that they accept CDRs that miss either or both the `start_date_time` and `end_date_time`, and local legislation allows billing of sessions where `start_date_time` and/or `end_date_time` are missing. Then, and only then, the CPO could send a CDR where the `start_date_time` and/or `end_date_time` are set to: "1970-1-1T00:00:00Z.

10.3.1.1. Example of a CDR


```
{
  "country_code": "BE",
  "party_id": "BEC",
  "id": "12345",
  "start_date_time": "2024-12-05T17:39:09Z",
  "end_date_time": "2024-12-05T19:37:32Z",
  "cdr_token": {
    "country_code": "DE",
    "party_id": "TNM",
    "uid": "012345678",
    "type": "RFID",
    "contract_id": "DE8ACC12E46L89"
  },
  "auth_method": "WHITELIST",
  "cdr_location": {
    "id": "LOC1",
    "name": "Gent Zuid",
    "address": "F.Rooseveltlaan 3A",
    "city": "Gent",
    "postal_code": "9000",
    "country": "BEL",
    "coordinates": {
      "latitude": "3.729944",
      "longitude": "51.047599"
    },
    "evse_uid": "3256",
    "evse_id": "BE*BEC*E041503003",
    "connector_id": "1",
    "connector_standard": "IEC_62196_T2",
    "connector_format": "SOCKET",
    "connector_power_type": "AC_1_PHASE"
  },
  "currency": "EUR",
  "tariffs": [{
    "country_code": "BE",
    "party_id": "BEC",
    "id": "12",
    "currency": "EUR",
    "elements": [{
      "price_components": [{
        "type": "TIME",
        "price": 2.00,
        "vat": 10.0,
        "step_size": 300
      }]
    }],
    "last_updated": "2024-05-02T14:15:01Z"
  }],
  "charging_periods": [{
    "start_date_time": "2024-12-05T17:39:09Z",
    "dimensions": [{
      "type": "TIME",
      "volume": 1.973
    }],
    "tariff_id": "12"
  }],
  "total_cost": {
    "before_taxes": 4.00,
```

```
      "taxes": [{
        "name": "VAT",
        "amount": 0.40
      }]
    },
    "total_energy": 15.342,
    "total_time": 1.973,
    "total_time_cost": {
      "before_taxes": 4.00,
      "taxes": [{
        "name": "VAT",
        "amount": 0.40
      }]
    },
    "last_updated": "2024-12-05T20:02:13Z"
  }
```


10.4. Data types

10.4.1. AuthMethod *enum*

Value	Description
AUTH_REQUEST	Authentication request has been sent to the eMSP.
COMMAND	Command like StartSession or ReserveNow used to start the Session, the Token provided in the Command was used as authorization.
WHITELIST	Whitelist used for authentication, no request to the eMSP has been performed.

10.4.2. CdrDimension *class*

Property	Type	Card.	Description
type	CdrDimensionType	1	Type of CDR dimension.
volume	number	1	Volume of the dimension consumed, measured according to the dimension type.

10.4.3. CdrDimensionType *enum*

This enumeration contains allowed values for CdrDimensions, which are used to define dimensions of ChargingPeriods in both `CDRs` and `Sessions`. Some of these values are not useful for `CDRs`, and SHALL therefor only be used in `Sessions`, these are marked in the column: Session Only

Value	Session Only	Description
CURRENT	Y	Average charging current during this ChargingPeriod: defined in A (Ampere). When negative, the current is flowing from the EV to the grid.

Value	Session Only	Description
ENERGY		Total amount of energy (dis-)charged during this ChargingPeriod: defined in kWh. When negative, more energy was feed into the grid then charged into the EV. Default step_size is 1.
ENERGY_EXPORT	Y	Total amount of energy feed back into the grid: defined in kWh.
ENERGY_IMPORT	Y	Total amount of energy charged, defined in kWh.
MAX_CURRENT		Sum of the maximum current over all phases, reached during this ChargingPeriod: defined in A (Ampere).
MIN_CURRENT		Sum of the minimum current over all phases, reached during this ChargingPeriod, when negative, current has flowed from the EV to the grid. Defined in A (Ampere).
MAX_POWER		Maximum power reached during this ChargingPeriod: defined in kW (Kilowatt).
MIN_POWER		Minimum power reached during this ChargingPeriod: defined in kW (Kilowatt), when negative, the power has flowed from the EV to the grid.
PARKING_TIME		Time during this ChargingPeriod not charging: defined in hours, default step_size multiplier is 1 second.
POWER	Y	Average power during this ChargingPeriod: defined in kW (Kilowatt). When negative, the power is flowing from the EV to the grid.
RESERVATION_TIME		Time during this ChargingPeriod Charge Point has been reserved and not yet been in use for this customer: defined in hours, default step_size multiplier is 1 second.
STATE_OF_CHARGE	Y	Current state of charge of the EV, in percentage, values allowed: 0 to 100. See note below.
TIME		Time charging during this ChargingPeriod: defined in hours, default step_size multiplier is 1 second.

NOTE

OCPI makes it possible to provide SoC in the Session object. This information can be useful to show the current State of Charge to an EV driver during charging. Implementers should be aware that SoC is only available at some DC Chargers. Which is currently a small amount of the total amount of Charge Points. Of these DC Chargers, only a small percentage currently provides SoC via OCPP to the CPO. Then there is also the question if SoC is allowed to be provided to third-parties as it can be seen as privacy-sensitive information. So if an implementer wants to show SoC in, for example an App, care should be taken, to make the App work without SoC, as this will probably not always be available.

10.4.4. CdrLocation *class*

The *CdrLocation* class contains only the relevant information from the Location object that is needed in a CDR.

Property	Type	Card.	Description
id	CiString(36)	1	Uniquely identifies the location within the CPO's platform (and suboperator platforms). This field can never be changed, modified or renamed.
name	string(255)	?	Display name of the location.
address	string(45)	1	Street/block name and house number if available.
city	string(45)	1	City or town.
postal_code	string(10)	?	Postal code of the location, may only be omitted when the location has no postal code: in some countries charging locations at highways don't have postal codes.
state	string(20)	?	State only to be used when relevant.
country	string(3)	1	ISO 3166-1 alpha-3 code for the country of this location.
coordinates	GeoLocation	1	Coordinates of the location.
evse_uid	CiString(36)	1	Uniquely identifies the EVSE within the CPO's platform (and suboperator platforms). For example a database unique ID or the actual *EVSE ID*. This field can never be changed, modified or renamed. This is the *technical* identification of the EVSE, not to be used as *human readable* identification, use the field: `evse_id` for that. Allowed to be set to: `#NA` when this CDR is created for a reservation that never resulted in a charging session.
evse_id	CiString(48)	1	Compliant with the following specification for EVSE ID: "E-mobility ID-codes: the purpose of IDs, ID usage and ID format" (https://evroaming.org/contract-evse-ids/). Allowed to be set to: `#NA` when this CDR is created for a reservation that never resulted in a charging session.
connector_id	CiString(36)	1	Identifier of the connector within the EVSE. Allowed to be set to: `#NA` when this CDR is created for a reservation that never resulted in a charging session.
connector_standard	ConnectorType	1	The standard of the installed connector. When this CDR is created for a reservation that never resulted in a charging session, this field can be set to any value and should be ignored by the Receiver.
connector_format	ConnectorFormat	1	The format (socket/cable) of the installed connector. When this CDR is created for a reservation that never resulted in a charging session, this field can be set to any value and should be ignored by the Receiver.
connector_power_type	PowerType	1	When this CDR is created for a reservation that never resulted in a charging session, this field can be set to any value and should be ignored by the Receiver.

10.4.5. CdrToken *class*

Property	Type	Card.	Description
country_code	CiString(2)	1	ISO-3166 alpha-2 country code of the MSP that 'owns' this Token.
party_id	CiString(3)	1	ID of the eMSP that 'owns' this Token (following the ISO-15118 standard).
uid	CiString(36)	1	Unique ID by which this Token can be identified. This is the field used by the CPO's system (RFID reader on the Charge Point) to identify this token. Currently, in most cases: `type=RFID`, this is the RFID hidden ID as read by the RFID reader, but that is not a requirement. If this is a `type=APP_USER` Token, it will be a unique, by the eMSP, generated ID.
type	TokenType	1	Type of the token
contract_id	CiString(36)	1	Uniquely identifies the EV driver contract token within the eMSP's platform (and suboperator platforms). Recommended to follow the specification for eMA ID from "E-mobility ID-codes: the purpose of IDs, ID usage and ID format" (https://evroaming.org/contract-evse-ids/).

10.4.6. ChargingPeriod *class*

A Charging Period consists of a start timestamp and a list of possible values that influence this period, for example: amount of energy charged this period, maximum current during this period etc.

Property	Type	Card.	Description
start_date_time	DateTime	1	Start timestamp of the charging period. A period ends when the next period starts. The last period ends when the session ends.
dimensions	CdrDimension	+	List of relevant values for this charging period.
tariff_id	CiString(36)	?	Unique identifier of the Tariff that is relevant for this Charging Period. If not provided, no Tariff is relevant during this period.

10.4.7. SignedData *class*

This class contains all the information of the signed data. Which encoding method is used, if needed, the public key and a list of signed values.

Property	Type	Card.	Description
encoding_method	CiString(36)	1	The name of the encoding used in the SignedData field. This is the name given to the encoding by a company or group of companies. See note below.
encoding_method_version	int	?	Version of the EncodingMethod (when applicable)

Property	Type	Card.	Description
public_key	string(512)	?	Public key used to sign the data, base64 encoded.
signed_values	SignedValue	+	One or more signed values.
url	string(512)	?	URL that can be shown to an EV driver. This URL gives the EV driver the possibility to check the signed data from a charging session.

NOTE For the German Eichrecht, different solutions are used, all have (somewhat) different encodings. Below the table with known implementations and the contact information for more information.

Name	Description	Contact
OCMF	Proposed by SAFE	https://has-to-be.com
Alfen Eichrecht	Alfen Eichrecht encoding / implementation.	https://alfen.com/de/downloads
EDL40 E-Mobility Extension	eBee smart technologies implementation	https://www.ebee.berlin
EDL40 Mennekes	Mennekes implementation	

10.4.8. SignedValue *class*

This class contains the signed and the plain/unsigned data. By decoding the data, the receiver can check if the content has not been altered.

Property	Type	Card.	Description
nature	CiString(32)	1	Nature of the value, in other words, the event this value belongs to. Possible values at moment of writing: - Start (value at the start of the Session) - End (signed value at the end of the Session) - Intermediate (signed values take during the Session, after Start, before End) Others might be added later.
plain_data	string(512)	1	The un-encoded string of data. The format of the content depends on the EncodingMethod field.
signed_data	string(5000)	1	Blob of signed data, base64 encoded. The format of the content depends on the EncodingMethod field.

11. *Tariffs* module

Module Identifier: `tariffs`

Data owner: `CPO`

Type: Functional Module

The Tariffs module gives eMSPs information about the tariffs used by the CPO.

11.1. Flow and Lifecycle

11.1.1. Push model

When the CPO creates a new Tariff they push them to the eMSPs by calling the PUT method on the eMSPs Tariffs endpoint with the newly created Tariff object.

Any changes to the Tariff(s) in the CPO's system can be sent to the eMSPs systems by calling the PUT method on the eMSPs Tariffs endpoint with the updated Tariff object.

When the CPO deletes a Tariff, they will update the eMSPs systems by calling DELETE on the eMSPs Tariffs endpoint with the ID of the Tariff that was deleted.

When the CPO is not sure about the state or existence of a Tariff object in the system of an eMSP, the CPO can use a GET request to validate the Tariff object in the eMSP's system.

11.1.2. Pull model

eMSPs who do not support the Push model need to call GET on the CPO's Tariff endpoint to receive all Tariffs, replacing the current list of known Tariffs with the newly received list.

11.2. Interfaces and Endpoints

There is both a Sender and a Receiver interface for Tariffs. Advised is to use the push direction from Sender to Receiver during normal operation. The Sender interface is meant to be used when the connection between two parties is established to retrieve the current list of Tariffs objects, and when the Receiver is not 100% sure the Tariff cache is still up-to-date.

11.2.1. Sender Interface

Typically implemented by market roles like: CPO.

The Sender's Tariffs interface gives the Receiver the ability to request Tariffs information.

Method	Description
GET	Returns Tariff objects from the CPO, last updated between the `{date_from}` and `{date_to}` (paginated)
POST	n/a

Method	Description
PUT	n/a
PATCH	n/a
DELETE	n/a

11.2.1.1. GET Method

Fetch information about all Tariffs.

Endpoint structure definition:

`{tariffs_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/cpo/2.2.1/tariffs/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/tariffs/?offset=50`

`https://www.server.com/ocpi/2.2.1/tariffs/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/cpo/2.2.1/tariffs/?offset=50&limit=100`

Request Parameters

If additional parameters: `{date_from}` and/or `{date_to}` are provided, only Tariffs with `last_updated` between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned.

This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	no	Only return Tariffs that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	no	Only return Tariffs that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

Response Data

The endpoint returns an object with a list of valid Tariffs, the header will contain the pagination related headers.

Any older information that is not specified in the response is considered no longer valid. Each object must contain all required fields. Fields that are not specified may be considered as null values.

Type	Card.	Description
Tariff	*	List of all tariffs.

11.2.2. Receiver Interface

Typically implemented by market roles like: eMSP and NSP.

Tariffs are Client Owned Objects, so the endpoints need to contain the required extra fields: {party_id} and {country_code}.

Endpoint structure definition:

```
{tariffs_endpoint_url}/{country_code}/{party_id}/{tariff_id}
```

Example:

```
https://www.server.com/ocpi/cpo/2.2.1/tariffs/BE/BEC/12
```

Method	Description
GET	Retrieve a Tariff as it is stored in the eMSP's system.
POST	n/a
PUT	Push new/updated Tariff object to the eMSP.
PATCH	n/a
DELETE	Remove a Tariff object which is no longer in use and will not be used in future either.

11.2.2.1. GET Method

If the CPO wants to check the status of a Tariff in the eMSP's system, it might GET the object from the eMSP's system for validation purposes. After all, the CPO is the owner of the object, so it would be illogical if the eMSP's system had a different status or was missing the object entirely.

Request Parameters

The following parameters SHALL be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO performing the GET request on the eMSP's system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO performing the GET request on the eMSP's system.
tariff_id	CiString(36)	yes	Tariff.id of the Tariff object to retrieve.

Response Data

The response contains the requested object.

Type	Card.	Description
Tariff	1	The requested Tariff object.

11.2.2.2. PUT Method

New or updated Tariff objects are pushed from the CPO to the eMSP.

Request Body

In the PUT request, the new or updated Tariff object is sent in the body.

Type	Card.	Description
Tariff	1	New or updated Tariff object.

Request Parameters

The following parameters SHALL be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO performing the PUT request on the eMSP's system. This SHALL be the same value as the `country_code` in the Tariff object being pushed.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO performing the PUT request on the eMSP's system. This SHALL be the same value as the `party_id` in the Tariff object being pushed.
tariff_id	CiString(36)	yes	Tariff.id of the Tariff object to create or replace.

Example: New Tariff € 2 per hour charging time (not parking).

```
PUT To URL: https://www.server.com/ocpi/emsp/2.2.1/tariffs/NL/TNM/12

{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "12",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "vat": 10.0,
      "step_size": 300
    }]
  }],
  "tax_included": "NO"
```

```
}
```

11.2.2.3. DELETE Method

Delete a Tariff object which is not used any more and will not be used in the future.

NOTE	Before deleting a Tariff object, it is RECOMMENDED to ensure that the Tariff object is not referenced by any Connector object within the `tariff_ids`.

Request Parameters

The following parameters SHALL be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO performing the PUT request on the eMSP's system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO performing the PUT request on the eMSP's system.
tariff_id	CiString(36)	yes	Tariff.id of the Tariff object to delete.

11.3. Object description

11.3.1. *Tariff* Object

A Tariff object consists of a list of one or more Tariff Elements, which in turn consist of Price Components.

A Tariff Element is a group of Price Components that apply under the same conditions. The rules for the conditions under which a Tariff Element applies are known as its "restrictions".

A Price Component describes how the usage of a particular dimension (time, energy, etcetera) is mapped to an amount of money owed.

This system of Tariffs, Tariff Elements and Price Components can be used to create complex Tariff structures.

When the list of Tariff Elements contains more than one Element that has a Price Component for a certain dimension, then the first Tariff Element with a Price Component for that dimension in the list with matching Tariff Restrictions will be used. Only one Price Component per dimension can be active at any point in time, but multiple Price Components for different dimensions can be active at once. That is you can have an ENERGY component and a TIME component active at the same time, but only those ones that are in the first Tariff Element that has a Price Component for that dimension and that has restrictions that match at that time.

When no Tariff Element with a specific Dimension is found for which the Restrictions match, and there is no Tariff Element in the list with the given Dimension without Restrictions, there will be no costs for that Tariff Dimension.

It is advised to always add a "default" Price Component per dimension.

This can be achieved by adding a Tariff Element without restrictions after all other occurrences of the same dimension in the list of Tariff Elements.

Such a Tariff Element will act as fallback when there is no other Tariff Element that has matching restrictions and that contains a Price Component for that dimension.

To define a "Free of Charge" tariff in OCPI, a Tariff containing one Tariff Element with no restrictions containing one Price Component with `type` = `FLAT` and `price` = `0.00` has to be provided.

See: Free of Charge Tariff example

NOTE

There are no parameters related to price rounding in the Tariff object or any of it constituent objects. Nor does the specification text of this module give any requirements about how to do price rounding. The reason for this that price rounding has to be done according to rules and restrictions set by applicable laws, contracts between the parties using OCPI and the currency used. The OCPI specification stays out of these matters.

Property	Type	Card.	Description
country_code	CiString(2)	1	ISO-3166 alpha-2 country code of the CPO that *owns* this Tariff.
party_id	CiString(3)	1	ID of the CPO that 'owns' this Tariff (following the ISO-15118 standard).
id	CiString(36)	1	Uniquely identifies the tariff within the CPO's platform (and suboperator platforms).
currency	string(3)	1	ISO-4217 code of the currency of this tariff.
type	TariffType	?	Defines the type of the tariff. This allows for distinction in case of given Charging Preferences. When omitted, this tariff is valid for all sessions.
tariff_alt_text	DisplayText	*	List of multi-language alternative tariff info texts.
tariff_alt_url	URL	?	URL to a web page that contains an explanation of the tariff information in human readable form.
min_price	PriceLimit	?	When this field is set, a Charging Session with this tariff will at least cost this amount.
max_price	PriceLimit	?	When this field is set, a Charging Session with this tariff will at most cost this amount.
preauthorize_amount	number	?	The amount that a Payment Terminal Provider should preauthorize when handling card payment for a Session with this Tariff.
elements	TariffElement	+	List of Tariff Elements.
tax_included	TaxIncluded	1	Whether taxes are included in the amounts in this Tariff.
start_date_time	DateTime	?	The time when this tariff becomes active, in UTC, `time_zone` field of the Location can be used to convert to local time. Typically used for a new tariff that is already given with the location, before it becomes active. (See note below)

Property	Type	Card.	Description
end_date_time	DateTime	?	The time after which this tariff is no longer valid, in UTC, time_zone field if the Location can be used to convert to local time. Typically used when this tariff is going to be replaced with a different tariff in the near future. (See note below)
energy_mix	EnergyMix	?	Details on the energy supplied with this tariff.
last_updated	DateTime	1	Timestamp when this Tariff was last updated (or created).

NOTE

min_price: As the taxes on a Charging Session might be different for different parts of the Session, there might be situations where the minimum cost after taxes is reached earlier or later than the minimum price before taxes. So as a rule, they both apply:

- The total cost of a Charging Session before taxes can never be lower than the value of the min_price's before_taxes field.
- The total cost of a Charging Session after taxes can never be lower than the value of the min_price's after_taxes field.

NOTE

max_price: As the taxes on a Charging Session might be different for different parts of the Session, there might be situations where the maximum cost after taxes is reached earlier or later than the maximum price before taxes. So as a rule, they both apply:

- The total cost of a Charging Session before taxes can never be higher than the value of the max_price's before_taxes field.
- The total cost of a Charging Session after taxes can never be higher than the value of the max_price's after_taxes field.

NOTE

start_date_time and end_date_time: When the Tariff of a Charge Point (Location) is changed during an ongoing charging session, it is common to not switch the Tariff until the ongoing session is finished. But this is NOT a requirement of OCPI, it is even possible with OCPI. Changing tariffs during an ongoing session is in many countries not allowed by consumer legislation. When charging at a Charge Point, a driver accepts the tariff which is valid when they start their charging session. If the Tariff of the Charge Point would change during the charging session, the driver might get billed something they didn't agree to when starting the session.

NOTE

The fields: tariff_alt_text and tariff_alt_url may be used separately, or in combination with each other or even combined with the structured list of Tariff Elements. When a Tariff contains both the tariff_alt_text and elements fields, the tariff_alt_text SHALL only contain additional tariff information in human-readable text, not the price information that is also available via the elements field. The reason for this is that the eMSP might have additional fees they want to include in communication with their customer.

11.3.1.1. Examples

In the following section, a few different pricing strategies will be explained with some Tariff examples. For simplicity, we will use the euro as the currency in all of the examples if not mentioned otherwise.

Simple Tariff example € 0.25 per kWh

- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 1 Wh

This tariff will result in costs of € 5.00 (excl. VAT) or € 5.50 (incl. VAT) when 20 kWh are charged.

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "16",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2018-12-17T11:16:55Z"
}
```

Tariff example € 0.25 per kWh + start fee

- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 1 Wh

This tariff will result in total cost of € 5.50 (excl. VAT) or € 6.10 (incl. VAT) when 20 kWh are charged.

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "17",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
```

```
  "tax_included": "NO",
  "last_updated": "2018-12-17T11:36:01Z"
}
```

Tariff example € 0.25 per kWh + minimum price

- Minimum price
  - € 0.50 (excl. VAT)
  - € 0.55 (incl. VAT, which is 10%)
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 1 Wh

This tariff will result in costs of € 5.00 (excl. VAT) or € 5.50 (incl. VAT) when 20 kWh are charged. But if less than 2 kWh is charged, € 0.50 (excl. VAT) or € 0.55 (incl. VAT) will be billed.

This is different from a start fee as can be seen when compared to the example above.

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "20",
  "currency": "EUR",
  "min_price": {
    "before_taxes": 0.50,
    "after_taxes": 0.55
  },
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2018-12-17T16:45:21Z"
}
```

Tariff example € 0.25 per kWh + parking fee + start fee

- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 1 Wh
- Parking

- € 2.00 per hour (excl. VAT)
- 20% VAT
- Billed per 15 min (900 seconds)

For a charging session where 20 kWh are charged and the vehicle is parked for 40 minutes after the session ended, this tariff will result in costs of € 7.00 (excl. VAT) or € 7.90 (incl. VAT). Because the parking time is billed per 15 minutes, the driver has to pay for 45 minutes of parking even though they left 40 minutes after their vehicle stopped charging.


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "18",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }, {
      "type": "PARKING_TIME",
      "price": 2.00,
      "vat": 20.0,
      "step_size": 900
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2018-12-17T11:44:10Z"
}
```


Tariff example € 0.25 per kWh + start fee + max price + tariff end date

- Maximum price
  - € 10 (excl. VAT)
  - € 11 (incl. VAT, which is 10%)
- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 1 Wh

This tariff has an end date: 30 June 2019, which is typically used when a tariff is going to be replaced by a new tariff. A Connector of a Location can have multiple Tariffs (IDs) assigned. By assigning both, the old and the new tariff ID, they will automatically be replaced. It is not required to update all Locations at the same time, the old tariff can also be removed later.

For a charging session where 50 kWh are charged, this tariff will result in costs of € 10.00 (excl. VAT) or € 11.00 (incl. VAT) due to the price limit. If only 30 kWh were charged, the costs would be € 8.00 (excl. VAT) and € 8.85 (incl. VAT), as the start fee combined with the energy costs would be lower than the defined max price.


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "16",
  "currency": "EUR",
  "max_price": {
    "before_taxes": 10.00,
    "after_taxes": 11.00
  },
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "end_date_time": "2019-06-30T23:59:59Z",
  "last_updated": "2018-12-17T17:15:01Z"
}
```


Simple Tariff example € 2 per hour

An example of a tariff where the driver does not pay per kWh, but for the time of using the Charge Point.

- Charging Time
  - € 2.00 per hour (excl. VAT)
  - 10% VAT
  - Billed per 1 min (60 seconds)

As this is tariff only has a `TIME` price_component, the driver will not be billed for time they are not charging: `PARKING_TIME`

For a charging session of 2.5 hours, this tariff will result in costs of € 5.00 (excl. VAT) or € 5.50 (incl. VAT).


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "12",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "vat": 10.0,
      "step_size": 60
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2015-06-29T20:39:09Z"
```

```
}
```

Simple Tariff with North American taxes

This is an example of how to represent Tariffs in Canada or the United States. In these countries, tax rates are not typically known beforehand to the CPO, so the `vat` field in the `PriceComponent` objects is not filled. The top level `tax_included` field in the Tariff object is used to say whether taxes are part of the prices in the Tariff, or if they will be added on top of those prices afterward.

This example Tariff is similar to the previous one in that it charges two currency units per hour of charging, but handles taxes in the North American way.

- Charging Time
  - C$ 2.00 per hour
  - Taxes not included
  - Billed per 1 min (60 seconds)

For a charging session of 2.5 hours, this tariff will result in costs of C$ 5.00, plus taxes according to locally applicable legislation.

```
{
  "country_code": "CA",
  "party_id": "FLO",
  "id": "12",
  "currency": "CAD",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2024-12-05T18:30:14Z"
}
```

Simple Tariff with North American taxes, price inclusive of tax

Sometimes, under North American style tax systems, Parties want to give prices including tax in their Tariffs. This example shows how to accomplish this.

- Charging Time
  - C$ 2.10 per hour
  - Taxes included
  - Billed per 1 min (60 seconds)

For a charging session of 2.5 hours, this tariff will result in costs of C$ 5.25. All taxes that are due are included in that C$5.25 amount.

```
{
  "country_code": "CA",
  "party_id": "FLO",
  "id": "12",
```

```
  "currency": "CAD",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.10,
      "step_size": 1
    }]
  }],
  "tax_included": "YES",
  "last_updated": "2024-12-05T18:30:14Z"
}
```

Simple Tariff example € 3 per hour, € 5 per hour parking

Example of a tariff where the driver pays for the time of using the Charge Point, but pays more when the car is no longer charging, to discourage the EV driver of leaving his EV connected when it is already full.

- Charging Time
  - € 3.00 per hour (excl. VAT)
  - 10% VAT
  - Billed per 1 min (60 seconds)
- Parking
  - € 5.00 per hour (excl. VAT)
  - 20% VAT
  - Billed per 5 min (300 seconds)

A charging session of 2.5 hours (charging), where the vehicle is parked for 42 more minutes after charging ended, results in a total session time of 150 minutes (charging) + 42 minutes (parking). This session with this tariff will result in total cost of € 11.25 (excl. VAT) or € 12.75 (incl. VAT). Because the parking time is billed per 5 minutes, the driver has to pay for 45 minutes of parking even though they left 42 minutes after their vehicle stopped charging.

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "21",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 3.00,
      "vat": 10.0,
      "step_size": 60
    }, {
      "type": "PARKING_TIME",
      "price": 5.00,
      "vat": 20.0,
      "step_size": 300
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2018-12-17T17:00:43Z"
}
```

Ad-Hoc simple Tariff example with multiple languages

For ad-hoc charging (paying for charging without a contract), the Tariff Elements are not as important. The eMSP is not involved when a driver uses ad-hoc payment at the Charge Point, so no CDR is sent to an eMSP. Having a good human readable text is much more useful.

- Charging Time
  - € 1.90 per hour (excl. VAT)
  - 5.2% VAT
  - Billed per 5 minutes (300 seconds)

For a charging session of 2.5 hours, this tariff will result in costs of € 4.75 (excl. VAT) or € 5.00 (incl. VAT).

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "12",
  "currency": "EUR",
  "type": "AD_HOC_PAYMENT",
  "tariff_alt_text": [{
    "language": "en",
    "text": "2.00 euro p/hour including VAT."
  }, {
    "language": "nl",
    "text": "2.00 euro p/uur inclusief BTW."
  }],
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 1.90,
      "vat": 5.2,
      "step_size": 300
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2015-06-29T20:39:09Z"
}
```

Ad-Hoc Tariff example not possible with OCPI

For this example, the credit card start tariff is € 0.50, but when using a debit card it is only € 0.25.

Such a tariff cannot be modeled with OCPI.
But by modeling it as € 0.50 start tariff where debit card users are given a discount in the final CDR of € 0.25, nobody is likely to complain. The `tariff_alt_text` explains this clearly.

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "19",
  "currency": "EUR",
  "type": "AD_HOC_PAYMENT",
  "tariff_alt_text": [{
    "language": "en",
    "text": "2.00 euro p/hour, start tariff debit card: 0.25 euro, credit card: 0.50 euro including VAT."
  }, {
    "language": "nl",
    "text": "2.00 euro p/uur, starttarief bankpas: 0,25 euro, creditkaart: 0,50 euro inclusief BTW."
  }],
```

```
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.40,
      "vat": 25.0,
      "step_size": 1
    }, {
      "type": "TIME",
      "price": 1.90,
      "vat": 5.2,
      "step_size": 300
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2018-12-29T15:55:58Z"
}
```

Simple Tariff example with alternative URL

This examples shows the use of `tariff_alt_url`.

This examples shows a `PROFILE_CHEAP` tariff, which is a smart charging tariff. Drivers are able to select this tariff by setting the `profile_type` in their Charging Preferences to `CHEAP`. In such case, the price might not be fixed, but depend on the real-time energy prices. To explain this to the driver, a short text inside `tariff_alt_text` might not be the best solution. Showing a graph could be better. Therefore it is also possible to provide an URL in `tariff_alt_url` to a site that explains the tariff better and in more detail.

- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 0.1 kWh (100 Wh)

For a charging session where 20.45 kWh are charged: this tariff will result in:

- Start fee: € 0.50 (excl. VAT), € 0.60 (incl. VAT)
- Energy costs: € 5.13 (excl. VAT), € 5.64 (incl. VAT)
- Total: € 5.63 (excl. VAT), € 6.24 (incl. VAT)

if the announced prices were billed. Because the energy is billed per 0.1 kWh, the driver has to pay for 20.5 kWh even though they only charged 20.45 kWh.

The twist here is that this tariff makes use of `tariff_alt_url` which links to a page with real-time energy prices of the operator, where is shown that the actual price per kWh is different. With an assumed current energy price of € 0.22 per kWh (excl. VAT), which is shown or explained on the page linked by `tariff_alt_url`, the resulting costs:

- Start fee: € 0.50 (excl. VAT), € 0.60 (incl. VAT)
- Energy costs: € 4.51 (excl. VAT), € 4.96 (incl. VAT)
- Total: € 5.01 (excl. VAT), € 5.56 (incl. VAT)

A breakdown for computing the price as the `elements` field of the Tariff says, with an energy price of € 0.25 / kWh, is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	0.50	0.50	20%	0.60
Energy	20.45 kWh	0.25 per kWh	5.11	10%	5.62
Total			5.61		6.22


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "13",
  "currency": "EUR",
  "type": "PROFILE_CHEAP",
  "tariff_alt_url": "https://company.com/tariffs/13",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 100
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2015-06-29T20:39:09Z"
}
```


Complex Tariff example

- Start or transaction fee
  - € 2.50 (excl. VAT)
  - 15% VAT
- Charging Time
  - When charging with less than 32A
    - € 1.00 per hour (excl. VAT)
    - 20% VAT
    - Billed per 15 min (900 seconds)
  - When charging with more than 32A on weekdays
    - € 2.00 per hour (excl. VAT)
    - 20% VAT
    - Billed per 10 min (600 seconds)
  - When charging with more than 32A on weekends
    - € 1.25 per hour (excl. VAT)

        - 20% VAT
        - Billed per 10 min (600 seconds)
- Parking
    - On weekdays between 09:00 and 18:00
        - € 5 per hour (excl. VAT)
        - 10% VAT
        - Billed per 5 min (300 seconds)
    - On Saturday between 10:00 and 17:00
        - € 6 per hour (excl. VAT)
        - 10% VAT
        - Billed per 5 min (300 seconds)

For a charging session on a Monday morning starting at 09:30 which takes 2:45 hours (165 minutes), where the driver uses a maximum of 16A of current and is parking for an additional 42 minutes afterwards, this tariff will result in costs of € 9.00 (excl. VAT) or € 10.30 (incl. VAT) for a total session time of 165 minutes (charging) + 42 minutes (parking).

A breakdown is as follows:

Dimension	**Quantity**	**Price ex VAT**	**Cost ex VAT**	**VAT**	**Cost inc VAT**
Flat	1	2.50	2.50	15%	2.875
Charging time	165 minutes	1.00 per hour	2.75	20%	3.30
Parking time	45 minutes	5.00 per hour	3.75	10%	4.125
Total			9.00		10.30

The step_size of the last time-based period is 5 so the parking time duration of 42 minutes is rounded up to 45. As such the driver has to pay for 45 minutes of parking while they were actually only parking for 42 minutes.

The charging time is not affected by step_size because it is followed by another time-based period.

For a charging session on a Saturday afternoon starting at 13:30 which takes 1:54 hours (114 minutes), where the driver uses a minimum of 43A of current (all the time, which is only theoretically possible) and is parking for an additional 71 minutes afterwards, this tariff will result in a total cost of € 12.28 (excl. VAT) or € 13.861 (incl. VAT). A breakdown is as follows:

Dimension	**Quantity**	**Price ex VAT**	**Cost ex VAT**	**VAT**	**Cost inc VAT**
Flat	1	2.50	2.50	15%	2.875
Charging time	114 minutes	1.25 per hour	2.28	20%	2.736
Parking time	75 minutes	6.00 per hour	7.50	10%	8.25
Total			12.28		13.861

The cost for parking time is 7.50, reflecting 75 minutes of parking, because the step_size of the last time-based period is applied to the 71 actual minutes of parking.

The charging time is again not affected by step_size because it is followed by parking time.


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "14",
  "currency": "EUR",
  "type": "REGULAR",
  "tariff_alt_url": "https://company.com/tariffs/14",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 2.50,
      "vat": 15.0,
      "step_size": 1
    }]
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 1.00,
      "vat": 20.0,
      "step_size": 900
    }],
    "restrictions": {
      "max_current": 32.00
    }
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "vat": 20.0,
      "step_size": 600
    }],
    "restrictions": {
      "min_current": 32.00,
      "day_of_week": ["MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"]
    }
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 1.25,
      "vat": 20.0,
      "step_size": 600
    }],
    "restrictions": {
      "min_current": 32.00,
      "day_of_week": ["SATURDAY", "SUNDAY"]
    }
  }, {
    "price_components": [{
      "type": "PARKING_TIME",
      "price": 5.00,
      "vat": 10.0,
      "step_size": 300
    }],
    "restrictions": {
      "start_time": "09:00",
      "end_time": "18:00",
      "day_of_week": ["MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"]
    }
  }, {
    "price_components": [{
      "type": "PARKING_TIME",
      "price": 6.00,
      "vat": 10.0,
```

```
      "step_size": 300
    }],
    "restrictions": {
      "start_time": "10:00",
      "end_time": "17:00",
      "day_of_week": ["SATURDAY"]
    }
  }],
  "tax_included": "NO",
  "last_updated": "2015-06-29T20:39:09Z"
}
```

Free of Charge Tariff example

In this example no VAT is given because it is not necessary (as the `price` is `0.00`). This might not always be the case though and it is of course permitted to add a VAT, even if the `price` is set to zero.

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "15",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.00,
      "step_size": 0
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2015-06-29T20:39:09Z"
}
```

Tariff example with reservation price

- Reservation
  - € 5.00 per hour (excl. VAT)
  - 20% VAT
  - Billed per 1 min (60 seconds)
- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 1 Wh

For a charging session that was started 15 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 6.75 (excl. VAT) or € 7.60 (incl. VAT).

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Reservation	15 minutes	5.00 per hour	1.25	20%	1.50
Total			6.75		7.60


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "20",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 5.00,
      "vat": 20.0,
      "step_size": 60
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2019-02-03T17:00:11Z"
}
```


Tariff example with reservation price and fee

- Reservation
  - € 2.00 reservation fee (excl. VAT)
  - € 5.00 per hour (excl. VAT)
  - 20% VAT
  - Billed per 5 min (300 seconds)
- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)

- 10% VAT
- Billed per 1 Wh

For a charging session that was started 13 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 8.75 (excl. VAT) or € 10.00 (incl. VAT). Because the reservation fee is billed per 5 minutes, the driver has to pay for 15 minutes of reservation even though they started the charging session 13 minutes after the reservation time.

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	2.00	2.00	20%	2.40
Parking time	15 minutes	5.00 per hour	1.25	20%	1.50
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Total			8.75		10.00


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "20",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 2.00,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "TIME",
      "price": 5.00,
      "vat": 20.0,
      "step_size": 300
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2019-02-03T17:00:11Z"
}
```

Tariff example with reservation price and expire fee

- Reservation
  - € 4.00 reservation expiration fee (excl. VAT) (*billed when a reservation expires and is not followed by a charging session*)
  - € 2.00 per hour (excl. VAT)
  - 20% VAT
  - Billed per 10 min (600 seconds)
- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
  - Billed per 1 Wh

This example is very similar to Tariff example with reservation price with the difference that expired reservations cost something and that reservation time is billed per 10 minutes. Also, the price for reservation is different.

For a charging session that was started 22 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 6.50 (excl. VAT) or € 7.30 (incl. VAT). Because the reservation fee is billed per 10 minutes, the driver has to pay for 30 minutes of reservation even though they started the charging session 22 minutes after the reservation time.

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time	30 minutes	2.00 per hour	1.00	20%	1.20
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Total			6.50		7.30

If the driver did not start a charging session and the reservation expired after the reserved time of 1 hour, the tariff would have resulted in costs of € 6.00 (excl. VAT) or € 7.20 (incl. VAT). In case a reservation is not used, the driver has to pay the full amount of reserved time as well as an additional expiration fee as compensation for not charging at all.

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	4.00	4.00	20%	4.80

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time	60 minutes	2.00 per hour	2.00	20%	2.40
Total			6.00		7.20


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "20",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 4.00,
      "vat": 20.0,
      "step_size": 1
    }],
    "restrictions": {
      "reservation": "RESERVATION_EXPIRES"
    }
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "vat": 20.0,
      "step_size": 600
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2019-02-03T17:00:11Z"
}
```


Tariff example with reservation time and expire time

- Reservation
  - € 3.00 per hour (excl. VAT)
  - € 6.00 per hour (excl. VAT) (*billed when a reservation expires and is not followed by a charging session*)
  - 20% VAT
  - Billed per 10 min (600 seconds)
- Start or transaction fee
  - € 0.50 (excl. VAT)

    - 20% VAT
- Energy
    - € 0.25 per kWh (excl. VAT)
    - 10% VAT
    - Billed per 1 Wh

This example is very similar to Tariff example with reservation price with the difference that expired reservations cost something and that reservation time is billed per 10 minutes. Also, the price for reservation is different.

For a charging session that was started 22 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 7.00 (excl. VAT) or € 7.90 (incl. VAT). Because the reservation fee is billed per 10 minutes, the driver has to pay for 30 minutes of reservation even though they started the charging session 22 minutes after the reservation time.

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time	30 minutes	3.00 per hour	1.50	20%	1.80
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Total			7.00		7.90

If the driver did not start a charging session and the reservation expired after the reserved time of 1.5 hours, the tariff would have resulted in costs of € 9.00 (excl. VAT) or € 10.80 (incl. VAT). In case a reservation is not used, the driver has to pay the expiration fee as compensation for not charging at all.

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time	90 minutes	6.00 per hour	9.00	20%	10.80
Total			9.00		10.80


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "20",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 6.00,
      "vat": 20.0,
      "step_size": 600
    }],
    "restrictions": {
      "reservation": "RESERVATION_EXPIRES"
    }
  }, {
```

```
    "price_components": [{
      "type": "TIME",
      "price": 3.00,
      "vat": 20.0,
      "step_size": 600
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2019-02-03T17:00:11Z"
}
```


11.4. Data types

11.4.1. DayOfWeek *enum*

Value	Description
MONDAY	Monday
TUESDAY	Tuesday
WEDNESDAY	Wednesday
THURSDAY	Thursday
FRIDAY	Friday
SATURDAY	Saturday
SUNDAY	Sunday

11.4.2. PriceComponent *class*

A Price Component describes how a certain amount of a certain dimension being consumed translates into an amount of money owed.

Property	Type	Card.	Description
type	TariffDimensionType	1	The dimension that is being priced
price	number	1	Price per unit for this dimension. This is including or excluding taxes according to the `tax_included` field of the Tariff that this PriceComponent is contained in.

Property	Type	Card.	Description
vat	number	?	Applicable VAT percentage for this tariff dimension. If omitted, no VAT is applicable.
step_size	int	1	Minimum amount to be billed. That is, the dimension will be billed in this step_size blocks. Consumed amounts are rounded up to the smallest multiple of step_size that is greater than the consumed amount. For example: if type is TIME and step_size has a value of 300, then time will be billed in blocks of 5 minutes. If 6 minutes were consumed, 10 minutes (2 blocks of step_size) will be billed.

NOTE

The step_size field is no longer present in OCPI 3.0. In OCPI 3.0, Parties are advised to measure quantities as precise as required by calibration law and use the full precision of such measurements in cost computation. Users of OCPI 2.2.1 looking to be ready for a transition to OCPI 3.0 or to maximize interoperability with OCPI 3.0 are advised to effectively avoid using step_size by setting step_size to 1 always.

NOTE

step_size: depends on the type and every type (except FLAT) defines a step_size multiplier, which is the size of every *step* for that type in the given unit.

For example: PARKING_TIME has the step_size multiplier: *1 second*, which means that the step_size of a Price Component is multiplied by *1 second*. Thus a step_size = 300 means 300 seconds (5 minutes). This means that when someone parked for 8 minutes they will be billed for 10 minutes. The parking time will be simply rounded up to the next larger chunk of step_size (i.e. blocks of 300 seconds in this example).

Another example: ENERGY has the step_size multiplied: *1 Wh*, which means that the step_size of a Price Component is multiplied by *1 Wh*. Thus a step_size = 1 with a price = 0.25 will result in a cost calculation that uses the charged Wh as precision.
If someone charges their EV with 115.2 Wh, then they are billed for 116 Wh, resulting in total cost of € 0.029.
When step_size = 25, then the same amount would be billed for 101 to 125 Wh: € 0.031.
When step_size = 500, then the same amount will be billed for 1 to 500 Wh: € 0.125.

NOTE

For more information about how step_size impacts the calculation of the cost of charging see: CDR object description

NOTE

Take into account that using step_size can be confusing for Drivers and other people. There may be local or national regulations that regulate step_size. For example in The Netherlands telecom companies are required to at least offer one subscription which is paid per second. To prevent confusion by the customer, we recommend to keep the step_size as small as possible and mention them clearly in your offering.

11.4.2.1. Example Tariff

Example Tariff to explain the step_size when switching from one Tariff Element to another:

- Charging fee of € 1.20 per hour (excl. VAT) before 17:00 with a `step_size` of 30 minutes (1800 seconds)
- Charging fee of € 2.40 per hour (excl. VAT) after 17:00 with a `step_size` of 15 minutes (900 seconds)
- Parking fee of € 1.00 per hour (excl. VAT) before 20:00 with a `step_size` of 15 minutes (900 seconds)


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "22",
  "currency": "EUR",
  "elements": [
    {
      "price_components": [
        {
          "type": "TIME",
          "price": 1.20,
          "step_size": 1800
        },
        {
          "type": "PARKING_TIME",
          "price": 1.00,
          "step_size": 900
        }
      ],
      "restrictions" : {
        "start_time" : "00:00",
        "end_time" : "17:00"
      }
    },
    {
      "price_components": [
        {
          "type": "TIME",
          "price": 2.40,
          "step_size": 900
        },
        {
          "type": "PARKING_TIME",
          "price": 1.00,
          "step_size": 900
        }
      ],
      "restrictions" : {
        "start_time" : "17:00",
        "end_time" : "20:00"
      }
    },
    {
      "price_components": [
        {
          "type": "TIME",
          "price": 2.40,
          "step_size": 900
        }
      ],
      "restrictions" : {
        "start_time" : "20:00",
        "end_time" : "00:00"
      }
    }
  ],
  "tax_included": "NO",
  "last_updated": "2018-12-18T17:07:11Z"
}
```

Example: switching to different Tariff Element #1

An EV driver plugs in at 16:55 and charges for 10 minutes (`TIME`). They then stop charging but stay plugged in for 2 more minutes (`PARKING_TIME`). The total session time is therefore 12 minutes. The parking time of 2 minutes is rounded to 15 minutes according to the step size of the last parking time period.

As a result,t he session costs € 0.55 ex VAT.

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT
Charging time	5 minutes	1.20 per hour	0.10
Charging time	5 minutes	2.40 per hour	0.20
Time	15 minutes	1.00 per hour	0.25
Total			0.55

Example: switching to different Tariff Element #2

An EV driver plugs in at 16:35 and charges for 35 minutes (`TIME`). After that they immediately unplug and leave without parking time.

As the charging time Price Component of the last Tariff Element being used has a `step_size` of 15 minutes, the total charging time is rounded up from 35 to 45 minutes. When considering the already billed 25 minutes of charging time before 17:00, we are left with 20 minutes to bill after 17:00.

That leads to a session fee of € 1.30. A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT
Charging time	25 minutes	1.20 per hour	0.50
Charging time	20 minutes	2.40 per hour	0.80
Total			1.30

Example: switching to Free-of-Charge Tariff Element

When parking becomes free after 20:00, there will not be an active `PARKING_TIME` Price Component nor a `TIME` Price Component. So the last parking period that needs to be paid, which is before 20:00, will be billed according to the `step_size` of the `PARKING_TIME PriceComponent` before 20:00.

An EV driver plugs in at 19:40 and charges for 12 minutes (`TIME`). They then stop charging but stay plugged in for 20 more minutes (`PARKING_TIME`). The total session time is therefore 32 minutes.

The total of billable parking time for the session is 8 minutes. This is rounded up to 15 minutes according to the step_size of the last time based Price Component that was active during the session. The extra 7 minutes are then added to the last period with a Price Component with a time-based dimension, that is the one from 19:52 to 20:00. So the user is billed € 0.60 for 15 minutes of parking and that makes a total session fee of € 0.80.

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT
Charging time	12 minutes	1.20 per hour	0.20
Time	15 minutes	2.40 per hour	0.60
Total			0.80

11.4.3. PriceLimit *class*

Property	Type	Card.	Description
before_taxes	number	1	Maximum or minimum cost excluding taxes.
after_taxes	number	?	Maximum or minimum cost including taxes.

11.4.4. ReservationRestrictionType *enum*

Value	Description
RESERVATION	Used in Tariff Elements to describe costs for a reservation.
RESERVATION_EXPIRES	Used in Tariff Elements to describe costs for a reservation that expires (i.e. driver does not start a charging session before expiry_date of the reservation).

NOTE When a Tariff has both `RESERVATION` and `RESERVATION_EXPIRES` Tariff Elements, where both Tariff Elements have a TIME Price Component, then the time based cost of an expired reservation will be calculated based on the `RESERVATION_EXPIRES` Tariff Element.

11.4.5. TariffElement *class*

A Tariff Element is a group of Price Components that share a set of restrictions under which they apply.

That the Price Components share the same restrictions does not mean that at any time, they either all apply or all do not apply. The reason is that applicable Price Components are looked up separately for each dimension, as described under the Tariff object. Therefore it is possible that a Price Component for one dimension is found in a Tariff Element that occurs earlier in the list of Tariff Elements than for another dimension.

Property	Type	Card.	Description
price_components	PriceComponent	+	List of Price Components that each describe how a certain dimension is priced.
restrictions	TariffRestrictions	?	Restrictions that describe under which circumstances the Price Components of this Tariff Element apply.

11.4.6. TariffDimensionType *enum*

Value	Description
ENERGY	Defined in kWh, `step_size` multiplier: 1 Wh
FLAT	Flat fee without unit for `step_size`
PARKING_TIME	Time not charging: defined in hours, `step_size` multiplier: 1 second
TIME	Time charging: defined in hours, `step_size` multiplier: 1 second Can also be used in combination with a RESERVATION restriction to describe the price of the reservation time.

11.4.7. TariffRestrictions *class*

```
A `TariffRestrictions` object describes if and when a Tariff Element becomes active or inactive during a Charging
Session.
```

These restrictions are not to be interpreted as making the Tariff Element applicable or not applicable for the entire Charging Session.

When more than one restriction is set, they are to be treated as a logical AND. So a Tariff Element is active if and only if all of the properties in its `TariffRestrictions` match.

Property	Type	Card.	Description
start_time	string(5)	?	Start time of day in local time, the time zone is defined in the `time_zone` field of the Location, for example 13:30, valid from this time of the day. Must be in 24h format with leading zeros. Hour/Minute separator: ":" Regex: `([0-1][0-9]|2[0-3]):[0-5][0-9]`
end_time	string(5)	?	End time of day in local time, the time zone is defined in the `time_zone` field of the Location, for example 19:45, valid until this time of the day. Same syntax as `start_time`. If end_time < start_time then the period wraps around to the next day. To stop at end of the day use: 00:00.
start_date	string(10)	?	Start date in local time, the time zone is defined in the `time_zone` field of the Location, for example: 2015-12-24, valid from this day (inclusive). Regex: `([12][0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])`
end_date	string(10)	?	End date in local time, the time zone is defined in the `time_zone` field of the Location, for example: 2015-12-27, valid until this day (exclusive). Same syntax as `start_date`.
min_kwh	number	?	Minimum consumed energy in kWh, for example 20, valid from this amount of energy (inclusive) being used.
max_kwh	number	?	Maximum consumed energy in kWh, for example 50, valid until this amount of energy (exclusive) being used.

Property	Type	Card.	Description
min_current	number	?	Sum of the minimum current (in Amperes) over all phases, for example 5. When the EV is charging with more than, or equal to, the defined amount of current, this TariffElement is/becomes active. If the charging current is or becomes lower, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the minimum current over the entire Charging Session. This restriction can make a TariffElement become active when the charging current is above the defined value, but the TariffElement MUST no longer be active when the charging current drops below the defined value.
max_current	number	?	Sum of the maximum current (in Amperes) over all phases, for example 20. When the EV is charging with less than the defined amount of current, this TariffElement becomes/is active. If the charging current is or becomes higher, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the maximum current over the entire Charging Session. This restriction can make a TariffElement become active when the charging current is below this value, but the TariffElement MUST no longer be active when the charging current raises above the defined value.
min_power	number	?	Minimum power in kW, for example 5. When the EV is charging with more than, or equal to, the defined amount of power, this TariffElement is/becomes active. If the charging power is or becomes lower, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the minimum power over the entire Charging Session. This restriction can make a TariffElement become active when the charging power is above this value, but the TariffElement MUST no longer be active when the charging power drops below the defined value.
max_power	number	?	Maximum power in kW, for example 20. When the EV is charging with less than the defined amount of power, this TariffElement becomes/is active. If the charging power is or becomes higher, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the maximum power over the entire Charging Session. This restriction can make a TariffElement become active when the charging power is below this value, but the TariffElement MUST no longer be active when the charging power raises above the defined value.
min_duration	int	?	Minimum duration in seconds the Charging Session MUST last (inclusive). When the duration of a Charging Session is longer than the defined value, this TariffElement is or becomes active. Before that moment, this TariffElement is not yet active.
max_duration	int	?	Maximum duration in seconds the Charging Session MUST last (exclusive). When the duration of a Charging Session is shorter than the defined value, this TariffElement is or becomes active. After that moment, this TariffElement is no longer active.
day_of_week	DayOfWeek	*	Which day(s) of the week this TariffElement is active.

Property	Type	Card.	Description
reservation	ReservationRestrictionType	?	When this field is present, the TariffElement describes reservation costs. A reservation starts when the reservation is made, and ends when the driver starts charging on the reserved EVSE/Location, or when the reservation expires. A reservation can only have: FLAT and TIME TariffDimensions, where TIME is for the duration of the reservation.

11.4.7.1. Example: Tariff with max_power Tariff Restrictions

Example Tariff to explain the `max_power` Tariff Restriction:

- Charging fee of € 0.20 per kWh (excl. VAT) when charging with a power of less than 16 kW.
- Charging fee of € 0.35 per kWh (excl. VAT) when charging with a power between 16 and 32 kW.
- Charging fee of € 0.50 per kWh (excl. VAT) when charging with a power above 32 kW (implemented as fallback tariff without Restriction).

For a charging session where the EV charges the first kWh with a power of 6 kW, increases the power to 48 kW for the next 40 kWh and reduces it again to 4 kW after that for another 0.5 kWh (probably due to physical limitations, i.e. temperature of the battery), this tariff will result in costs of € 20.30 (excl. VAT). The costs are composed of the following components:

- 1 kWh at 6 kW: € 0.20
- 40 kWh at 48 kW: € 20.00
- 0.5 kWh at 4 kW: € 0.10


```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "1",
  "currency": "EUR",
  "type": "REGULAR",
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.20,
      "vat": 20.0,
      "step_size": 1
    }],
    "restrictions": {
      "max_power": 16.00
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
      "price": 0.35,
      "vat": 20.0,
      "step_size": 1
    }],
    "restrictions": {
      "max_power": 32.00
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
```

```
      "price": 0.50,
      "vat": 20.0,
      "step_size": 1
    }]
  }],
  "tax_included": "NO",
  "last_updated": "2018-12-05T12:01:09Z"
}
```

11.4.7.2. Example: Tariff with max_duration Tariff Restrictions

A supermarket wants to allow their customer to charge for free. As most customers will be out of the store in 20 minutes, they allow free charging for 30 minutes. If a customer charges longer than that, they will charge them the normal price per kWh. But as they want to discourage long usage of their Charge Points, charging becomes much more expensive after 1 hour:

- First 30 minutes of charging is free.
- Charging fee of € 0.25 per kWh (excl. VAT) after 30 minutes.
- Charging fee of € 0.40 per kWh (excl. VAT) after 60 minutes.

For a charging session with a duration of 40 minutes where 5 kWh are charged during the first 30 minutes and another 1.2 kWh in the remaining 10 minutes of the session, this tariff will result in costs of € 0.30 (excl. VAT). The costs are composed of the following components:

- 5 kWh for free: € 0.00
- 1.2 kWh at 0.25/kWh: € 0.30

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "2",
  "currency": "EUR",
  "type": "REGULAR",
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.00,
      "vat": 20.0,
      "step_size": 1
    }],
    "restrictions": {
      "max_duration": 1800
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
      "price": 0.25,
      "vat": 20.0,
      "step_size": 1
    }],
    "restrictions": {
      "max_duration": 3600
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
      "price": 0.40,
      "vat": 20.0,
      "step_size": 1
    }]
```

```
  }],
  "tax_included": "NO",
  "last_updated": "2018-12-05T13:12:44Z"
}
```

11.4.8. TariffType *enum*

Value	Description
AD_HOC_PAYMENT	Used to describe that a Tariff is valid when ad-hoc payment is used at the Charge Point (for example: Debit or Credit card payment terminal).
PROFILE_CHEAP	Used to describe that a Tariff is valid when Charging Preference: CHEAP is set for the session.
PROFILE_FAST	Used to describe that a Tariff is valid when Charging Preference: FAST is set for the session.
PROFILE_GREEN	Used to describe that a Tariff is valid when Charging Preference: GREEN is set for the session.
REGULAR	Used to describe that a Tariff is valid when using an RFID, without any Charging Preference, or when Charging Preference: REGULAR is set for the session.

11.4.9. TaxIncluded *enum*

Describes if tax may have to be added to the amounts in a Tariff.

Value	Description
YES	Taxes are included in the prices in this Tariff.
NO	Taxes are not included, and will be added on top of the prices in this Tariff.
N/A	No taxes are applicable to this Tariff.

12. *Tokens* module

Module Identifier: `tokens`

Data owner: `MSP`

Type: Functional Module

The tokens module gives CPOs knowledge of the token information of an eMSP. eMSPs can push Token information to CPOs, CPOs can build a cache of known Tokens. When a request to authorize comes from a Charge Point, the CPO can check against this cache. With this cached information they know to which eMSP they can later send a CDR.

12.1. Flow and Lifecycle

12.1.1. Push model

When the eMSP creates a new Token object they push it to the CPO by calling PUT on the CPO's Tokens endpoint with the newly created Token object.

Any changes to Token in the eMSP system are sent to the CPO system by calling either the PUT or the PATCH on the CPO's Tokens endpoint with the updated Token(s).

When the eMSP invalidates a Token (deleting is not possible), the eMSP will send the updated Token (with the field: valid set to `false`, by calling, either the PUT or the PATCH on the CPO's Tokens endpoint with the updated Token.

When the eMSP is not sure about the state or existence of a Token object in the CPO system, the eMSP can call the GET to validate the Token object in the CPO system.

12.1.2. Pull model

When a CPO is not sure about the state of the list of known Tokens, or wants to request the full list as a start-up of their system, the CPO can call the GET on the eMSP's Token endpoint to receive all Tokens, updating already known Tokens and adding new received Tokens to it own list of Tokens. This is not intended for real-time operation, requesting the full list of tokens for every authorization will put to much strain on systems. It is intended for getting in-sync with the server, or to get a list of all tokens (from a server without Push mode) every X hours.

12.1.3. Real-time authorization

An eMSP might want their Tokens to be authorized 'real-time', not white-listed. For this the eMSP has to implement the POST Authorize request and set the Token.whitelist field to `NEVER` for Tokens they want to have authorized 'real-time'.

If an eMSP doesn't want real-time authorization, the POST Authorize request doesn't have to be implemented as long as all their Tokens have Token.whitelist set to `ALWAYS`.

When an eMSP does not want to Push the full list of tokens to CPOs, the CPOs will need to call the POST Authorize request to check if a Token is known by the eMSP, and if it is valid.

NOTE	Doing real-time authorization of RFID will mean a longer delay of the authorization process, which might result in bad user experience at the Charge Point. So care should be taken to keep delays in

processing the request to an absolute minimum.

NOTE Real-time authorization might be asked for a charging location that is not published via the Location module, typically a private charger. In most cases this is expected to result in: `ALLOWED`.

NOTE If real-time authorization is asked for a location, the eMSP SHALL NOT validate that charging is possible based on information like opening hours or EVSE status etc. as this information might not be up to date.

12.2. Interfaces and endpoints

There is both a Sender and a Receiver interface for Tokens. It is advised to use the Push direction from Sender to Receiver during normal operation. The Sender interface is meant to be used when the Receiver is not 100% sure the Token cache is still correct.

12.2.1. Receiver Interface

Typically implemented by market roles like: CPO.

With this interface the Sender can push the Token information to the Receiver. Tokens is a Client Owned Object, so the end-points need to contain the required extra fields: {party_id} and {country_code}.

Endpoint structure definition:

```
{token_endpoint_url}/{country_code}/{party_id}/{token_uid}[?type={type}]
```

Example:

```
https://www.server.com/ocpi/cpo/2.2.1/tokens/NL/TNM/012345678
```

Method	Description
GET	Retrieve a Token as it is stored in the CPO system.
POST	n/a
PUT	Push new/updated Token object to the CPO.
PATCH	Notify the CPO of partial updates to a Token.
DELETE	n/a, (Use PUT, Tokens cannot be removed).

12.2.1.1. GET Method

If the eMSP wants to check the status of a Token in the CPO system it might GET the object from the CPO system for validation purposes. The eMSP is the owner of the objects, so it would be illogical if the CPO system had a different status or was missing an object.

Request Parameters

The following parameters: `country_code`, `party_id`, `token_uid` have to be provided as URL segments.

The parameter: `type` may be provided as an URL parameter

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the eMSP requesting this GET from the CPO system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the eMSP requesting this GET from the CPO system.
token_uid	CiString(36)	yes	Token.uid of the Token object to retrieve.
type	TokenType	no	Token.type of the Token to retrieve. Default if omitted: RFID

Response Data

The response contains the requested object.

Type	Card.	Description
Token	1	The requested Token object.

12.2.1.2. PUT Method

New or updated Token objects are pushed from the eMSP to the CPO.

Request Body

In the put request a new or updated Token object is sent.

Type	Card.	Description
Token	1	New or updated Token object.

Request Parameters

The following parameters: `country_code`, `party_id`, `token_uid` have to be provided as URL segments.

The parameter: `type` may be provided as an URL parameter

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the eMSP sending this PUT request to the CPO system. This SHALL be the same value as the `country_code` in the Token object being pushed.
party_id	CiString(3)	yes	Party ID (Provider ID) of the eMSP sending this PUT request to the CPO system. This SHALL be the same value as the `party_id` in the Token object being pushed.
token_uid	CiString(36)	yes	Token.uid of the (new) Token object (to replace).
type	TokenType	no	Token.type of the Token of the (new) Token object (to replace). Default if omitted: RFID

Example: put a new Token

```
PUT To URL: https://www.server.com/ocpi/cpo/2.2.1/tokens/NL/TNM/012345678

{
  "country_code": "NL",
  "party_id": "TNM",
  "uid": "012345678",
  "type": "RFID",
  "contract_id": "NL8ACC12E46L89",
  "visual_number": "DF000-2001-8999-1",
  "issuer": "TheNewMotion",
  "group_id": "DF000-2001-8999",
  "valid": true,
  "whitelist": "ALWAYS",
  "last_updated": "2015-06-29T22:39:09Z"
}
```

12.2.1.3. PATCH Method

Same as the PUT method, but only the fields/objects that have to be updated have to be present, other fields/objects that are not specified are considered unchanged.

Any request to the PATCH method SHALL contain the `last_updated` field.

Example: invalidate a Token

```
PATCH To URL: https://www.server.com/ocpi/cpo/2.2.1/tokens/NL/TNM/012345678

{
  "valid": false,
  "last_updated": "2019-06-19T02:11:11Z"
}
```

12.2.2. Sender Interface

Typically implemented by market roles like: eMSP.

This interface enables the Receiver to request the current list of Tokens, when needed. Via the POST method it is possible to authorize a single token.

Method	Description
GET	Get the list of known Tokens, last updated between the {date_from} and {date_to} (paginated)
POST	Real-time authorization request
PUT	n/a
PATCH	n/a
DELETE	n/a

12.2.2.1. GET Method

Fetch information about Tokens known in the eMSP systems.

Endpoint structure definition:

`{tokens_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/emsp/2.2.1/tokens/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/tokens/?offset=50`

`https://www.server.com/ocpi/2.2.1/tokens/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/emsp/2.2.1/tokens/?offset=50&limit=100`

Request Parameters

If additional parameters: `{date_from}` and/or `{date_to}` are provided, only Tokens with (`last_updated`) between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned.

This request is paginated, it supports the pagination related URL parameters. This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	no	Only return Tokens that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	no	Only return Tokens that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

Response Data

The endpoint response with list of valid Token objects, the header will contain the pagination related headers.

Any older information that is not specified in the response is considered as no longer valid. Each object must contain all required fields. Fields that are not specified may be considered as null values.

Type	Card.	Description
Token	*	List of all tokens.

12.2.2.2. POST Method

Do a 'real-time' authorization request to the eMSP system, validating if a Token might be used (at the optionally given Location).

Endpoint structure definition:

`{tokens_endpoint_url}{token_uid}/authorize[?type={type}]`

The `/authorize` is required for the real-time authorize request.

Examples:

`https://www.server.com/ocpi/emsp/2.2.1/tokens/012345678/authorize`

`https://ocpi.server.com/2.2.1/tokens/012345678/authorize?type=RFID`

When the eMSP does not know the Token, the eMSP SHALL respond with an HTTP status code: 404 (Not Found).

When the eMSP receives a 'real-time' authorization request from a CPO that contains too little information (no LocationReferences provided) to determine if the Token might be used, the eMSP SHALL respond with the OCPI status: 2002

Request Parameters

The parameter: `token_uid` has to be provided as URL segments.

The parameter: `type` may be provided as an URL parameter

Parameter	Datatype	Required	Description
token_uid	CiString(36)	yes	Token.uid of the Token for which authorization is requested.
type	TokenType	no	Token.type of the Token for which this authorization is. Default if omitted: RFID

Request Body

In the body an optional LocationReferences object can be given. The eMSP SHALL then validate if the Token is allowed to be used at this Location, and if applicable: which of the Locations EVSEs. The object with valid Location and EVSEs will be returned in the response.

Type	Card.	Description
LocationReferences	?	Location and EVSEs for which the Token is requested to be authorized.

Response Data

When the token is known by the Sender, the response SHALL contain a AuthorizationInfo object.

If the token is not known, the response SHALL contain the status code: `2004: Unknown Token`, and no `data` field.

Type	Card.	Description
AuthorizationInfo	1	Contains information about the authorization, if the Token is allowed to charge and optionally which EVSEs are allowed to be used.

12.3. Object description

12.3.1. *AuthorizationInfo* Object

Property	Type	Card.	Description
allowed	AllowedType	1	Status of the Token, and whether charging is allowed at the optionally given location.
token	Token	1	The complete Token object for which this authorization was requested.
location	LocationReferences	?	Optional reference to the location if it was included in the request, and if the EV driver is allowed to charge at that location. Only the EVSEs the EV driver is allowed to charge at are returned.
authorization_reference	CiString(36)	?	Reference to the authorization given by the eMSP, when given, this reference will be provided in the relevant Session and/or CDR.
info	DisplayText	?	Optional display text, additional information to the EV driver.

12.3.2. *Token* Object

Property	Type	Card.	Description
country_code	CiString(2)	1	ISO-3166 alpha-2 country code of the MSP that 'owns' this Token.
party_id	CiString(3)	1	ID of the eMSP that 'owns' this Token (following the ISO-15118 standard).
uid	CiString(36)	1	Unique ID by which this Token, combined with the Token type, can be identified. This is the field used by CPO system (RFID reader on the Charge Point) to identify this token. Currently, in most cases: type=RFID, this is the RFID hidden ID as read by the RFID reader, but that is not a requirement. If this is a `APP_USER` or `AD_HOC_USER` Token, it will be a uniquely, by the eMSP, generated ID. This field is named `uid` instead of `id` to prevent confusion with: `contract_id`.
type	TokenType	1	Type of the token
contract_id	CiString(36)	1	Uniquely identifies the EV Driver contract token within the eMSP's platform (and suboperator platforms). Recommended to follow the specification for eMA ID from"E-mobility ID-codes: the purpose of IDs, ID usage and ID format" (https://evroaming.org/contract-evse-ids/).
visual_number	string(64)	?	Visual readable number/identification as printed on the Token (RFID card), might be equal to the contract_id.

Property	Type	Card.	Description
issuer	string(64)	1	Issuing company, most of the times the name of the company printed on the token (RFID card), not necessarily the eMSP.
group_id	CiString(36)	?	This ID groups a couple of tokens. This can be used to make two or more tokens work as one, so that a session can be started with one token and stopped with another, handy when a card and key-fob are given to the EV-driver. Beware that OCPP 1.5/1.6 only support group_ids (it is called parentId in OCPP 1.5/1.6) with a maximum length of 20.
valid	boolean	1	Is this Token valid
whitelist	WhitelistType	1	Indicates what type of white-listing is allowed.
language	string(2)	?	Language Code ISO 639-1. This optional field indicates the Token owner's preferred interface language. If the language is not provided or not supported then the CPO is free to choose its own language.
default_profile_type	ProfileType	?	The default Charging Preference. When this is provided, and a charging session is started on an Charge Point that support Preference base Smart Charging and support this ProfileType, the Charge Point can start using this ProfileType, without this having to be set via: Set Charging Preferences.
energy_contract	EnergyContract	?	When the Charge Point supports using your own energy supplier/contract at a Charge Point, information about the energy supplier/contract is needed so the CPO knows which energy supplier to use. NOTE: In a lot of countries it is currently not allowed/possible to use a drivers own energy supplier/contract at a Charge Point.
last_updated	DateTime	1	Timestamp when this Token was last updated (or created).

The combination of *uid* and *type* should be unique for every token within the eMSP's system.

NOTE	OCPP supports group_id (or ParentID as it is called in OCPP 1.5/1.6) OCPP 1.5/1.6 only support group ID's with maximum length of string(20), case insensitive. As long as EV-driver can be expected to charge at an OCPP 1.5/1.6 Charge Point, it is adviced to not used a group_id longer then 20.

12.3.2.1. Examples

Simple APP_USER example

```
{
  "country_code": "DE",
  "party_id": "TNM",
  "uid": "bdf21bce-fc97-11e8-8eb2-f2801f1b9fd1",
  "type": "APP_USER",
  "contract_id": "DE8ACC12E46L89",
  "issuer": "TheNewMotion",
  "valid": true,
```

```
  "whitelist": "ALLOWED",
  "last_updated": "2018-12-10T17:16:15Z"
}
```


Full RFID example


```
{
  "country_code": "DE",
  "party_id": "TNM",
  "uid": "12345678905880",
  "type": "RFID",
  "contract_id": "DE8ACC12E46L89",
  "visual_number": "DF000-2001-8999-1",
  "issuer": "TheNewMotion",
  "group_id": "DF000-2001-8999",
  "valid": true,
  "whitelist": "ALLOWED",
  "language": "it",
  "default_profile_type": "GREEN",
  "energy_contract": {
    "supplier_name": "Greenpeace Energy eG",
    "contract_id": "0123456789"
  },
  "last_updated": "2018-12-10T17:25:10Z"
}
```


12.4. Data types

12.4.1. AllowedType *enum*

Value	Description
ALLOWED	This Token is allowed to charge (at this location).
BLOCKED	This Token is blocked.
EXPIRED	This Token has expired.
NO_CREDIT	This Token belongs to an account that has not enough credits to charge (at the given location).
NOT_ALLOWED	Token is valid, but is not allowed to charge at the given location.

12.4.2. EnergyContract *class*

Information about a energy contract that belongs to a Token so a driver could use his/her own energy contract when charging at a Charge Point.

Property	Type	Card.	Description
supplier_name	string(64)	1	Name of the energy supplier for this token.
contract_id	string(64)	?	Contract ID at the energy supplier, that belongs to the owner of this token.

12.4.3. LocationReferences *class*

References to location details.

Property	Type	Card.	Description
location_id	CiString(36)	1	Unique identifier for the location.
evse_uids	CiString(36)	*	Unique identifiers for EVSEs within the CPO's platform for the EVSE within the given location.

12.4.4. TokenType *OpenEnum*

Value	Description
AD_HOC_USER	One time use Token ID generated by a server (or App.) The eMSP uses this to bind a Session to a customer, probably an app user.
APP_USER	Token ID generated by a server (or App.) to identify a user of an App. The same user uses the same Token for every Session.
EMAID	An EMAID. EMAIDs are used as Tokens when the Charging Station and the vehicle are using ISO 15118 for communication.
OTHER	Other type of token
RFID	RFID Token

NOTE The eMSP is RECOMMENDED to push Tokens with type: `AD_HOC_USER` or `APP_USER` with `whitelist` set to `NEVER`. Whitelists are very useful for RFID type Tokens, but the `AD_HOC_USER`/`APP_USER` Tokens are used to start Sessions from an App etc. so whitelisting them has no advantages.

NOTE The eMSP is RECOMMENDED to not push Tokens with type `EMAID` at all. Exchanging Token objects for EMAID Tokens is not necessary because the CPO already learns which Party issued the Token from the Charging Station. The CPO can then contact this Party for real-time authorization using real-time authorization.

NOTE The management of the contract certificates that are used with ISO 15118 to authenticate the vehicle is left outside of OCPI 2.3.0. There are other existing standards for exchanging and validating certificates that Parties can use to authenticate contract certificates.

12.4.5. WhitelistType *enum*

Defines when authorization of a Token by the CPO is allowed.

The validity of a Token has no influence on this. If a Token is: `valid = false`, when the `whitelist` field requires real-time authorization, the CPO SHALL do a real-time authorization, the state of the Token might have changed.

Value	Description
ALWAYS	Token always has to be whitelisted, realtime authorization is not possible/allowed. CPO shall always allow any use of this Token.
ALLOWED	It is allowed to whitelist the token, realtime authorization is also allowed. The CPO may choose which version of authorization to use.
ALLOWED_OFFLINE	In normal situations realtime authorization shall be used. But when the CPO cannot get a response from the eMSP (communication between CPO and eMSP is offline), the CPO shall allow this Token to be used.
NEVER	Whitelisting is forbidden, only realtime authorization is allowed. CPO shall always send a realtime authorization for any use of this Token to the eMSP.

13. *Commands* module

Module Identifier: `commands`

Type: Functional Module

The Commands module enables remote commands to be sent to a Location/EVSE. The following commands are supported:

- `CANCEL_RESERVATION`
- `RESERVE_NOW`
- `START_SESSION`
- `STOP_SESSION`
- `UNLOCK_CONNECTOR`

See CommandType for a description of the different commands. *Use the* `UNLOCK_CONNECTOR` *command with care, please read the note at CommandType.*

Module dependency: Locations module, Sessions module

13.1. Flow

With the Commands module, commands can be sent from the eMSP, via the CPO to a Charge Point. Most Charge Points are hooked up to the internet via a relative slow wireless connection. To prevent long blocking calls, the commands module is designed to work asynchronously.

The Sender (typically eMSP) send a request to a Receiver (typically CPO), via the Receivers Commands interface. The Receiver checks if it can send the request to a Charge Point and will respond to the request with a status, indicating if the request can be sent to a Charge Point.

The Receiver (typically CPO) sends the requested command (via another protocol, for example: OCPP) to a Charge Point. The Charge Point will respond if it understands the command and will try to execute the command. This response doesn't always mean that the command was executed successfully. The Receiver (typically CPO) will forward the result in a new POST request to the Senders Commands interface.

The following examples try to give insight into the message flow and the asynchronous nature of the OCPI Commands.

Example of a `START_SESSION` that is accepted, but no new Session is started because EV not plugged in before end of time-out. This is an example for Charge Point that allows a remote start when the cable is not yet plugged in. Some Charge Points even require this, there might, for example, be a latch in front of the socket to prevent vandalism.

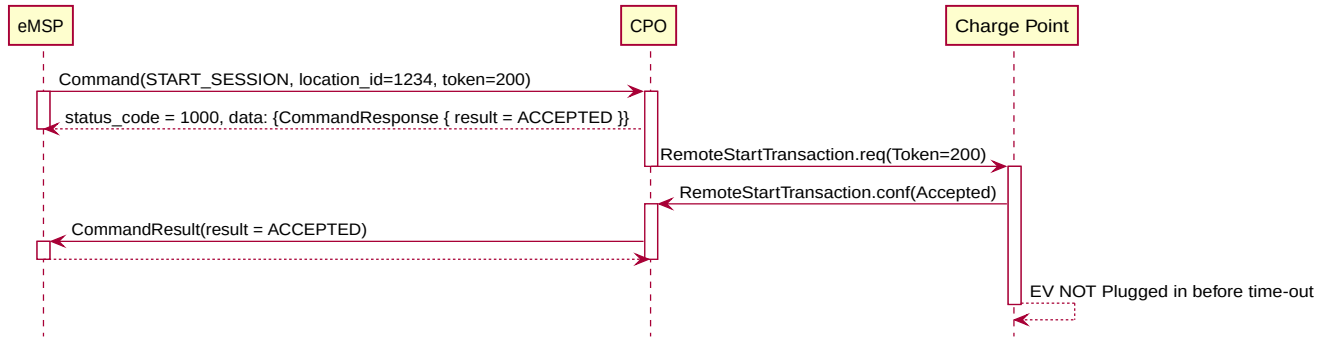


Figure 28. START_SESSION failed

Example of a START_SESSION that is accepted, but no new Session is started because the EV is not plugged in, and this Charge Point does not allow a remote start without a cable already being plugged in.

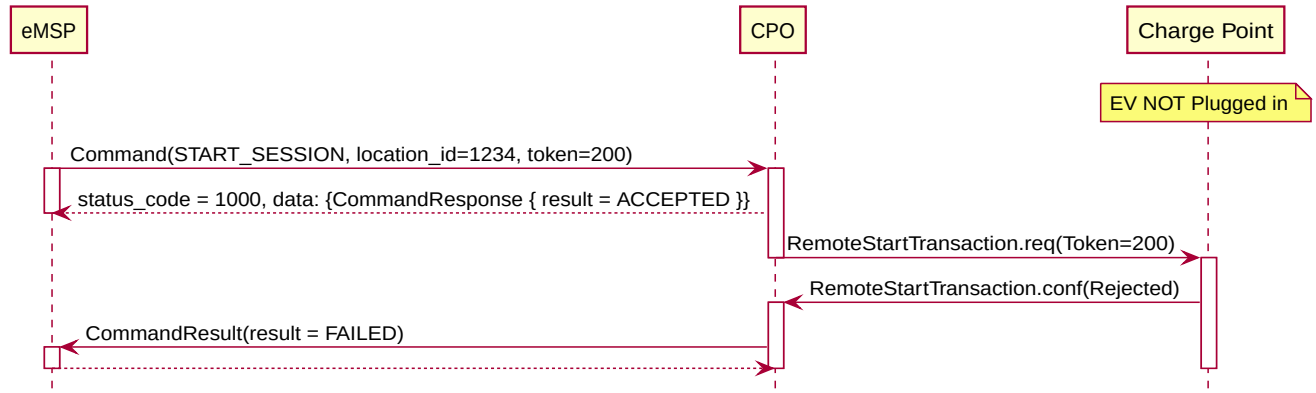


Figure 29. START_SESSION failed

Example of a START_SESSION that is accepted and results in a new Session.

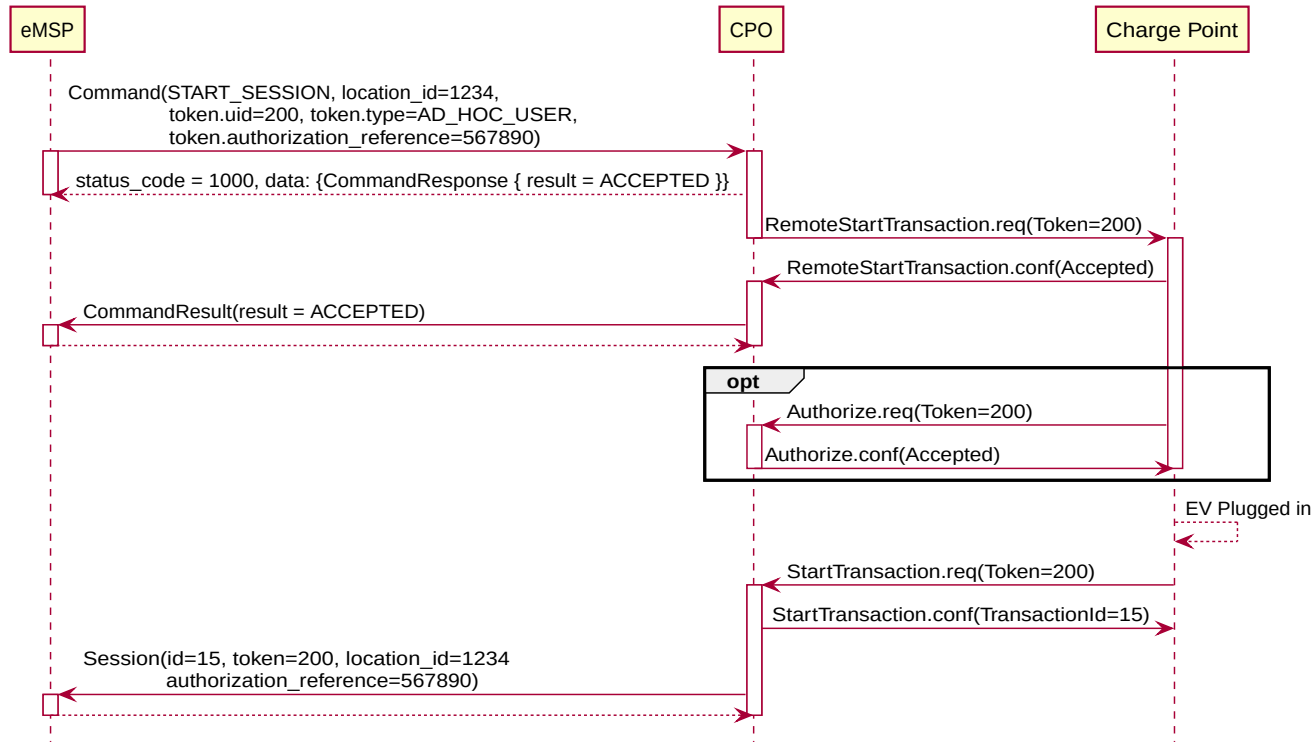


Figure 30. START_SESSION successful

Example of a START_SESSION with a Token that is Whitelist: NEVER.

The CPO should not check the Token in the START_SESSION, before sending it to the Charge Point. The CPO should assume that the eMSP only sends valid Tokens in the START_SESSION object.

If needed, the Charge Point does an OCPP Authorize request to validate the Token (proved via OCPP). In such case the CPO only does an realtime authorization when the OCPP Authorize request is for an RFID Token and the START_SESSION for this Token was received more then 15 minutes ago.

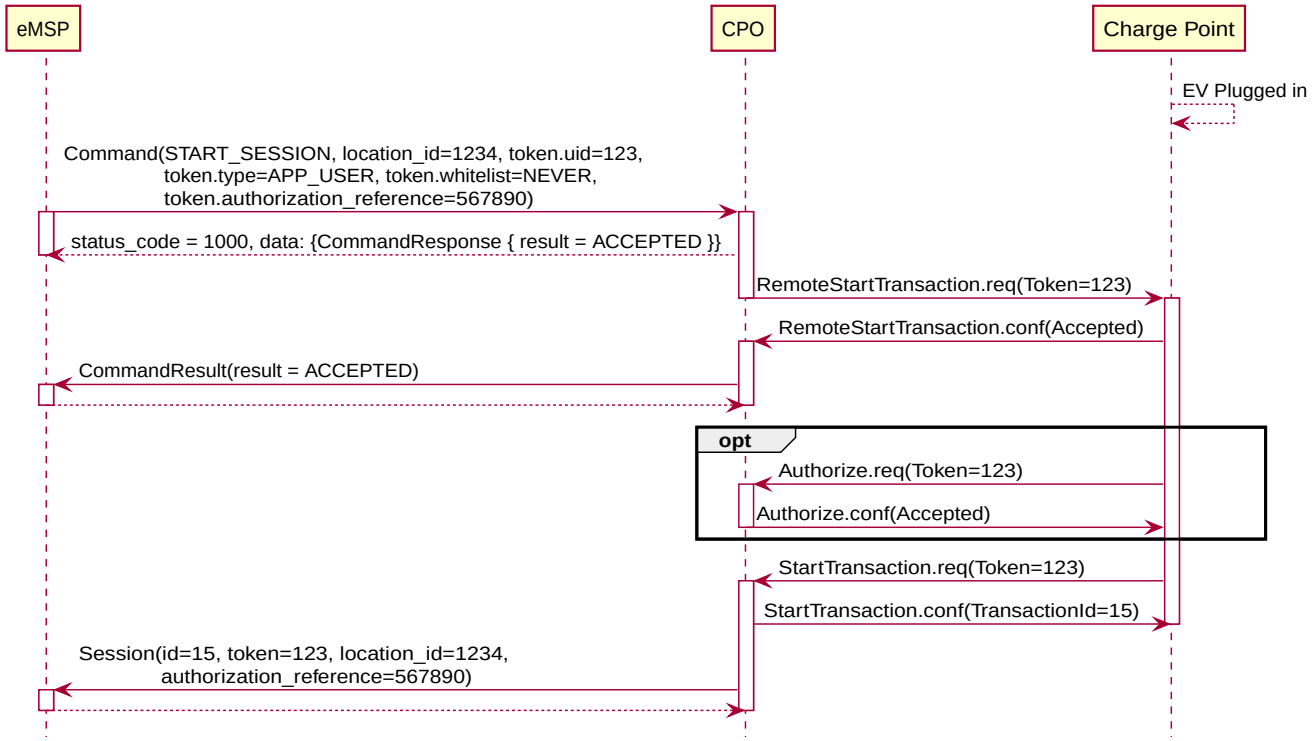


Figure 31. START_SESSION whitelist NEVER

Example of a `UNLOCK_CONNECTOR` that fails because the Location is not known by the CPO.

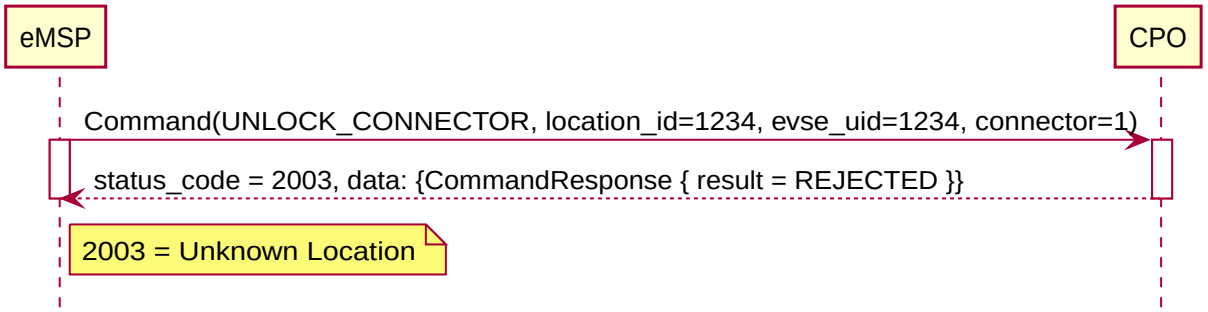


Figure 32. UNLOCK_CONNECTOR Unknown Location

Example of a `RESERVE_NOW` that is rejected by the Charge Point.

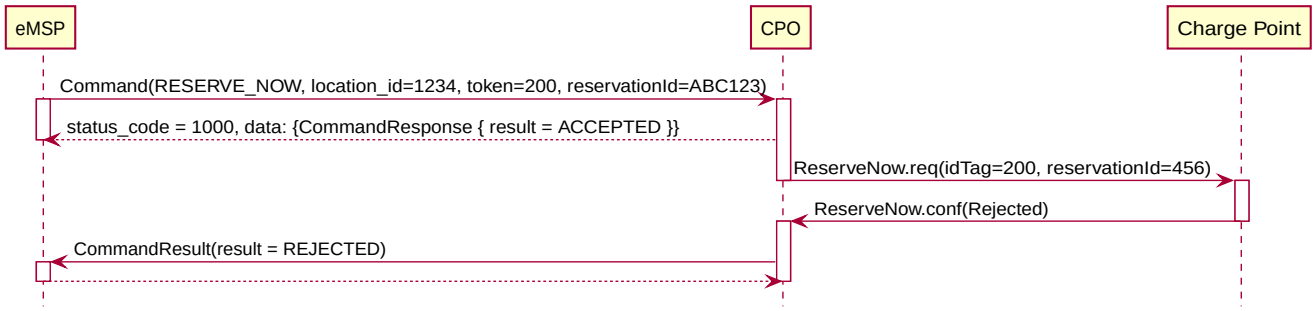


Figure 33. RESERVE_NOW rejected by Charge Point

Example of a successful `RESERVE_NOW`.

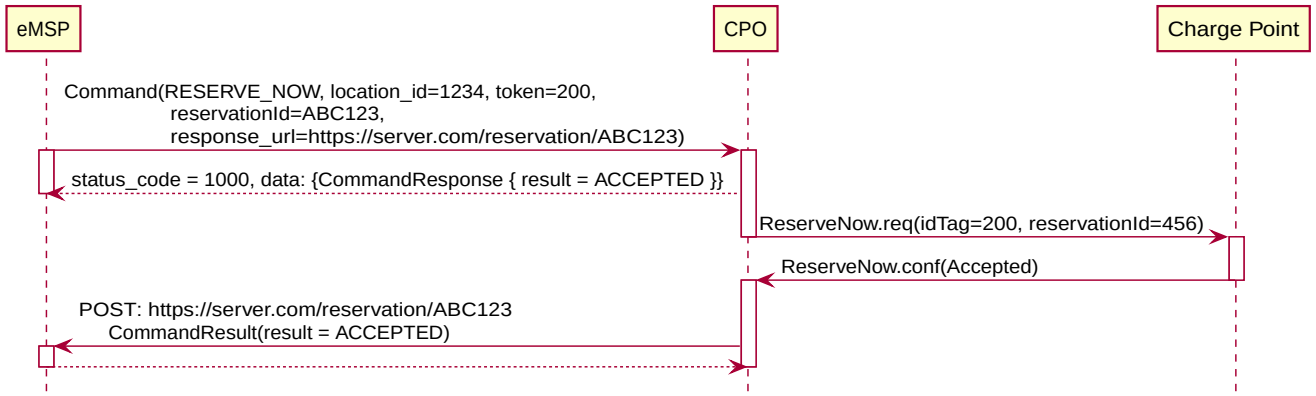


Figure 34. Successful RESERVE_NOW

Reservation canceled by the CPO.
OCPI makes it possible for a CPO to cancel a reservation. This is not to be taken lightly. When a driver makes a reservation of a Charge Point/EVSE, he/she wants to be sure to have a charging location. So if the CPO cancel the reservation, the driver will for sure not like it. But there are some circumstances where the CPO is forced to cancel a reservation. For example: Charge Point has become defect, or the CPO is notified of ongoing roadworks which makes the Charge Point unreachable etc.

To Cancel a reservation the CPO call the Senders interface with the same URL as was given by the Sender (eMSP) when the `RESERVE_NOW` command was send.

The sequence diagram below continues after the sequence diagram above.

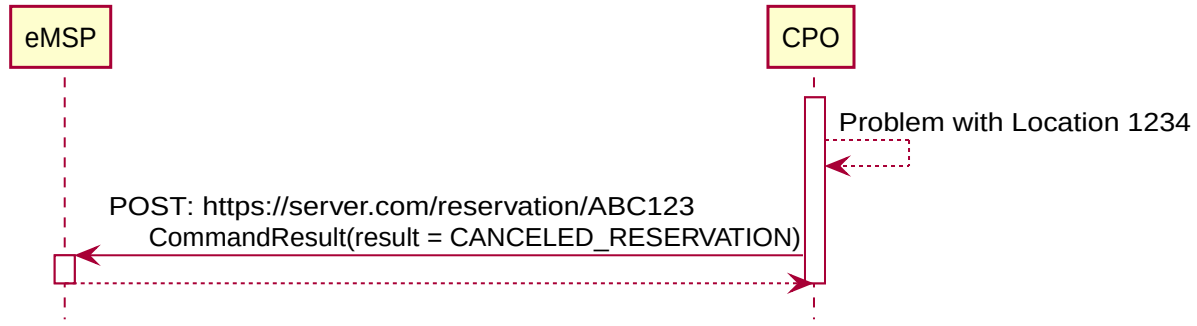


Figure 35. Reservation canceled by the CPO

These examples use OCPP 1.6 based commands between CPO and Charge Point, but that is not a requirement for OCPI.

If the Sender (typically eMSP) wants to have a reference between the calls sent to the Receivers interface and the

asynchronous result received from the Charge Point via the CPO, the Sender can make some unique identifier part of the`response_url` that is part of every method in the Receiver interface. The Receiver will call this URL when the result is received from the Charge Point. The Sender can then match the unique identifier from the URL called with the request.

13.2. Interfaces and endpoints

The commands module consists of two interfaces: a Receiver interface that enables a Sender (typically eMSP) (and its clients) to send commands to a Location/EVSE and an Sender interface to receive the response from the Location/EVSE asynchronously.

13.2.1. Receiver Interface

Typically implemented by market roles like: CPO.

Endpoint structure definition:

`{commands_endpoint_url}{command}`

Examples:

`https://www.server.com/ocpi/cpo/2.2.1/commands/START_SESSION`

`https://ocpi.server.com/commands/STOP_SESSION`

`https://server.com/ocpi/cpo/2.2.1/commands/RESERVE_NOW`

Method	Description
GET	n/a
POST	Send a command to the CPO, requesting the CPO to send the command to the Charge Point
PUT	n/a
PATCH	n/a
DELETE	n/a

13.2.1.1. POST Method

Request Parameters

The following parameter shall be provided as URL segments.

Parameter	Datatype	Required	Description
command	CommandType	yes	Type of command that is requested.

13.2.1.2. Request Body

Depending on the `command` parameter the body SHALL contain the applicable object for that command.

Type	Card.	Description
Choice: one of five		
> CancelReservation	1	CancelReservation object, for the `CANCEL_RESERVATION` command, with information needed to cancel an existing reservation.
> ReserveNow	1	ReserveNow object, for the `RESERVE_NOW` command, with information needed to reserve a (specific) connector of a Charge Point for a given Token.
> StartSession	1	StartSession object, for the `START_SESSION` command, with information needed to start a sessions.
> StopSession	1	StopSession object, for the `STOP_SESSION` command, with information needed to stop a sessions.
> UnlockConnector	1	UnlockConnector object, for the `UNLOCK_CONNECTOR` command, with information needed to unlock a connector of a Charge Point.

Response Data

The response contains the direct response from the Receiver, not the response from the Charge Point itself, that will be sent via an asynchronous POST on the Sender interface if this response is `ACCEPTED`.

Datatype	Card.	Description
CommandResponse	1	Result of the command request, by the CPO (not the Charge Point). So this indicates if the CPO understood the command request and was able to send it to the Charge Point. This is not the response by the Charge Point

13.2.2. Sender Interface

Typically implemented by market roles like: eMSP.

The Sender interface receives the asynchronous responses.

Endpoint structure definition:

No structure defined. This is open to the Sender to define, the URL is provided to the Receiver by the Sender in the POST to the Receiver interface. Therefor OCPI does not define variables.

Example:

`https://www.server.com/ocpi/emsp/2.2.1/commands/{command}`

`https://ocpi.server.com/commands/{command}/{uid}`

Method	Description
GET	n/a
POST	Receive the asynchronous response from the Charge Point.
PUT	n/a

Method	Description
PATCH	n/a
DELETE	n/a

13.2.2.1. POST Method

Endpoint structure definition:

It is up to the implementation of the eMSP to determine what parameters are put in the URL. The eMSP sends a URL in the POST method body to the CPO. The CPO is required to use this URL for the asynchronous response by the Charge Point. It is advised to make this URL unique for every request to differentiate simultaneous commands, for example by adding a unique id as a URL segment.

Examples:

`https://www.server.com/ocpi/emsp/2.2.1/commands/RESERVE_NOW/1234`

`https://www.server.com/ocpi/emsp/2.2.1/commands/UNLOCK_CONNECTOR/2`

13.2.2.2. Request Body

Datatype	Card.	Description
CommandResult	1	Result of the command request, from the Charge Point.

13.3. Object description

13.3.1. *CancelReservation* Object

With CancelReservation the Sender can request the Cancel of an existing Reservation. The CancelReservation needs to contain the `reservation_id` that was given by the Sender to the `ReserveNow`.

As there might be cost involved for a Reservation, canceling a reservation might still result in a CDR being send for the reservation.

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between CancelReservation requests.
reservation_id	CiString(36)	1	Reservation id, unique for this reservation. If the Charge Point already has a reservation that matches this reservationId the Charge Point will replace the reservation.

13.3.2. *CommandResponse* Object

The CommandResponse object is send in the HTTP response body.

Because OCPI does not allow/require retries, it could happen that the asynchronous result url given by the eMSP is never successfully called. The eMSP might have had a glitch, HTTP 500 returned, was offline for a moment etc. For the eMSP to be able to give a quick as possible response to another system or driver app. It is important for the eMSP to know the timeout on a certain command.

Property	Type	Card.	Description
result	CommandResponseType	1	Response from the CPO on the command request.
timeout	int	1	Timeout for this command in seconds. When the Result is not received within this timeout, the eMSP can assume that the message might never be send.
message	DisplayText	*	Human-readable description of the result (if one can be provided), multiple languages can be provided.

13.3.3. *CommandResult* Object

Property	Type	Card.	Description
result	CommandResultType	1	Result of the command request as sent by the Charge Point to the CPO.
message	DisplayText	*	Human-readable description of the reason (if one can be provided), multiple languages can be provided.

13.3.4. *ReserveNow* Object

The `evse_uid` is optional. If no EVSE is specified, the Charge Point should keep one EVSE available for the EV Driver identified by the given Token. (This might not be supported by all Charge Points). A reservation can be replaced/updated by sending a `RESERVE_NOW` request with the same Location (Charge Point) and the same `reservation_id`.

A successful reservation will result in a new `Session` object being created by the CPO.

An unused Reservation of a Charge Point/EVSE MAY result in cost being made, thus also a CDR.

The eMSP provides a Token that has to be used by the Charge Point. The Token provided by the eMSP for the `ReserveNow` SHALL be authorized by the eMSP before sending it to the CPO. Therefor the CPO SHALL NOT check the validity of the Token provided before sending the request to the Charge Point.

If this is an OCPP Charge Point, the Charge Point decides if it needs to validate the given Token, in such case:

- If this Token is of type `AD_HOC_USER` or `APP_USER` the CPO SHALL NOT do a realtime authorization at the eMSP for this.
- If this Token is of type `RFID`, the CPO SHALL NOT do a realtime authorization at the eMSP for this Token at the given EVSE/Charge Point within 15 minutes after having received this `ReserveNow`.

The eMSP MAY use Tokens that have not been pushed via the Token module. This is especially likely with tokens fof types `AD_HOC_USER` or `APP_USER`. Such Tokens are only used in commands sent by an eMSP and never presented locally

at the Charge Point by a Driver like RFID Tokens.

Unknown Tokens received by the CPO in the ReserveNow Object don't need to be stored in the Token module. In other words, when a Token has been received via ReserveNow, the same Token does not have to be returned in a Token GET request from the eMSP.

An eMSP sending a ReserveNow SHALL only use Tokens that are owned by this eMSP. Using Tokens of other eMSPs is not allowed.

The reservation_id sent by the Sender (eMSP) to the Receiver (CPO) SHALL NOT be sent directly to a Charge Point. The CPO SHALL make sure the Reservation ID sent to the Charge Point is unique and is not used by another Sender (eMSP). We don't want a Sender (eMSP) to replace or cancel a reservation of another Sender (eMSP).

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between ReserveNow requests.
token	Token	1	Token object for how to reserve this Charge Point (and specific EVSE).
expiry_date	DateTime	1	The Date/Time when this reservation ends, in UTC.
reservation_id	CiString(36)	1	Reservation id, unique for this reservation. If the Receiver (typically CPO) Point already has a reservation that matches this reservationId for that Location it will replace the reservation.
location_id	CiString(36)	1	Location.id of the Location (belonging to the CPO this request is sent to) for which to reserve an EVSE.
evse_uid	CiString(36)	?	Optional EVSE.uid of the EVSE of this Location if a specific EVSE has to be reserved.
authorization_reference	CiString(36)	?	Reference to the authorization given by the eMSP, when given, this reference will be provided in the relevant Session and/or CDR.

13.3.5. *StartSession* Object

The evse_uid is optional. If no EVSE is specified, the Charge Point can itself decide on which EVSE to start a new session. (this might not be supported by all Charge Points).

The eMSP provides a Token that has to be used by the Charge Point. The Token provided by the eMSP for the StartSession SHALL be authorized by the eMSP before sending it to the CPO. Therefor the CPO SHALL NOT check the validity of the Token provided before sending the request to the Charge Point.

If this is an OCPP Charge Point, the Charge Point decides if it needs to validate the given Token, in such case:

- If this Token is of type: AD_HOC_USER or APP_USER the CPO SHALL NOT do a realtime authorization at the eMSP for this .
- If this Token is of type: RFID, the CPO SHALL NOT do a realtime authorization at the eMSP for this Token at the

given EVSE/Charge Point within 15 minutes after having received this StartSession. (This means that if the driver decided to use his RFID within 15 minutes at the same Charge Point, because the app is not working somehow, the RFID is already authorized)

The eMSP MAY use Tokens that have not been pushed via the Token module, especially AD_HOC_USER or APP_USER Tokens are only used by commands send by an eMSP. As these are never used locally at the Charge Point like RFID.

Unknown Tokens received by the CPO in the StartSession Object don't need to be stored in the Token module. In other words, when a Token has been received via StartSession, the same Token does not have to be returned in a Token GET request from the eMSP. However, the information of the Token SHALL be put in the Session and CDR.

An eMSP sending a StartSession SHALL only use Token that are owned by this eMSP in StartSession, using Tokens of other eMSPs is not allowed.

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between StartSession requests.
token	Token	1	Token object the Charge Point has to use to start a new session. The Token provided in this request is authorized by the eMSP.
location_id	CiString(36)	1	Location.id of the Location (belonging to the CPO this request is sent to) on which a session is to be started.
evse_uid	CiString(36)	?	Optional EVSE.uid of the EVSE of this Location on which a session is to be started. Required when connector_id is set.
connector_id	CiString(36)	?	Optional Connector.id of the Connector of the EVSE on which a session is to be started. This field is required when the capability: START_SESSION_CONNECTOR_REQUIRED is set on the EVSE.
authorization_reference	CiString(36)	?	Reference to the authorization given by the eMSP, when given, this reference will be provided in the relevant Session and/or CDR.

NOTE In case of an OCPP 1.x Charge Point, the EVSE ID should be mapped to the connector ID of a Charge Point. OCPP 1.x does not have good support for Charge Points that have multiple connectors per EVSE. To make StartSession over OCPI work, the CPO SHOULD present the different connectors of an EVSE as separate EVSE, as is also written by the OCA in the application note: "Multiple Connectors per EVSE in a OCPP 1.x implementation".

13.3.6. *StopSession* Object

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between StopSession requests.

Property	Type	Card.	Description
session_id	CiString(36)	1	Session.id of the Session that is requested to be stopped.

13.3.7. *UnlockConnector* Object

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between UnlockConnector requests.
location_id	CiString(36)	1	Location.id of the Location (belonging to the CPO this request is sent to) of which it is requested to unlock the connector.
evse_uid	CiString(36)	1	EVSE.uid of the EVSE of this Location of which it is requested to unlock the connector.
connector_id	CiString(36)	1	Connector.id of the Connector of this Location of which it is requested to unlock.

13.4. Data types

13.4.1. CommandResponseType *enum*

Response to the command request from the eMSP to the CPO.

Value	Description
NOT_SUPPORTED	The requested command is not supported by this CPO, Charge Point, EVSE etc.
REJECTED	Command request rejected by the CPO. (Session might not be from a customer of the eMSP that send this request)
ACCEPTED	Command request accepted by the CPO.
UNKNOWN_SESSION	The Session in the requested command is not known by this CPO.

13.4.2. CommandResultType *enum*

Result of the command that was sent to the Charge Point.

Value	Description
ACCEPTED	Command request accepted by the Charge Point.
CANCELED_RESERVATION	The Reservation has been canceled by the CPO.
EVSE_OCCUPIED	EVSE is currently occupied, another session is ongoing. Cannot start a new session
EVSE_INOPERATIVE	EVSE is currently inoperative or faulted.

Value	Description
FAILED	Execution of the command failed at the Charge Point.
NOT_SUPPORTED	The requested command is not supported by this Charge Point, EVSE etc.
REJECTED	Command request rejected by the Charge Point.
TIMEOUT	Command request timeout, no response received from the Charge Point in a reasonable time.
UNKNOWN_RESERVATION	The Reservation in the requested command is not known by this Charge Point.

13.4.3. CommandType *OpenEnum*

The command requested.

Value	Description
CANCEL_RESERVATION	Request the Charge Point to cancel a specific reservation.
RESERVE_NOW	Request the Charge Point to reserve a (specific) EVSE for a Token for a certain time, starting now.
START_SESSION	Request the Charge Point to start a transaction on the given EVSE/Connector.
STOP_SESSION	Request the Charge Point to stop an ongoing session.
UNLOCK_CONNECTOR	Request the Charge Point to unlock the connector (if applicable). This functionality is for help desk operators only!

The command `UNLOCK_CONNECTOR` may only be used by an operator or the eMSP. This command SHALL never be allowed to be sent directly by the EV-Driver. The `UNLOCK_CONNECTOR` is intended to be used in the rare situation that the connector is not unlocked successfully after a transaction is stopped. The mechanical unlock of the lock mechanism might get stuck, for example: fail when there is tension on the charging cable when the Charge Point tries to unlock the connector. In such a situation the EV-Driver can call either the CPO or the eMSP to retry the unlocking.

14. *ChargingProfiles* module

Module Identifier: `chargingprofiles`

Type: Functional Module

With the ChargingProfiles module, parties (SCSP but also MSPs) can send (Smart) Charging Profiles to a Location/EVSE. It is also possible to request the 'ActiveChargingProfile' from a Location/EVSE.

The ActiveChargingProfile is the charging profile as calculated by the EVSE. It is the result of the calculation of all smart charging inputs present in the EVSE, also Local Limits might be taken into account.

The ChargingProfile is similar to the concept of Charging Profiles in OCPP, but exposes this functionality to third parties. These objects and the accompanying interfaces make certain abstractions that make them more suitable for energy parties to signal their intent. The data structures are base on OCPP 1.6 and 2.0 to make conversion of messages between OCPI and OCPP easy.

NOTE	Charging Profiles set via this module are no guarantee that the EV will charge with the exact given limit, it is a maximum limit, not a target. A lot of factors influence the charging speed. The EV might not take the amount of energy that the EVSE is willing to provide to it, the battery might be too warm or almost full. A single phase cable might be used on a three phase Charge Point. There can be local energy limits (load balancing between EVSEs on a relative small energy connection to a group of EVSEs) that might limit the energy offered by the EVSE to the EV even further.

ChargingProfile can be created by the owner of a Token on Sessions that belong to that token. If another party sends a ChargingProfile and the CPO has no contract that allows that party to set profiles on sessions, the CPO is allowed to reject such profiles.

This module can be used by the eMSP, but can also be used by another party that provide "Smart Charging Services" (Smart Charging Service Provider (SCSP) / Aggregator / Energy Service Broker etc.) These SCSPs then depend on the CPO sending session information to them. They need to know which session is ongoing to be able to influence it. If a SCSP uses this module, read eMSP as SCSP.

NOTE	OCPI provides the means for SCSPs to do this. Parties doing this have to adhere to local privacy laws, have to have setup contracts etc. Local laws might oblige explicit consent from the driver etc.

Module dependency: Sessions module

14.1. Smart Charging Topologies

There are different Smart Charging Topologies possible. Which topology can be used depends on the contracts between different parties.

NOTE	Care has to be taken to prevent mixing the different topologies. When multiple parties start sending Charging Profiles, the resulting charging speed might be unpredictable. In case of OCPP Charge Points, the result will be the minimum of all the Charging Profiles, resulting in a slower than needed charging speed.

14.1.1. The eMSP generates ChargingProfiles.

The most straight forward topology, the eMSP generates ChargingProfiles for its own customers, no SCSP is involved. The eMSP 'owns' the customer, so if the eMSP knows that its customer agrees with the eMSP manipulating the charging speed, the eMSP is free to do this.

Figure 36. Smart Charging Topology: The eMSP generates ChargingProfiles.

Interface	Role
Sender	eMSP
Receiver	CPO

14.1.2. The eMSP delegated Smart Charging to SCSP.

In the topology, the eMSP has delegated the generation of ChargingProfiles to a SCSP. For this, the eMSP and SCSP have agreed to use OCPI as the interface.

The eMSP 'owns' the customer, so if the eMSP knows that its customer agrees with the eMSP manipulating the charging speed, the eMSP is free to do this. The eMSP can forward OCPI Session Objects to the SCSP. the SCSP can act on the received/updated Session Objects, by sending Charging Profile commands via the eMSP to the CPO.

The eMSP and SCSP have to take into account that they have to oblige to local privacy laws when exchanging information about eMSPs customers.

From the CPO point of view, this topology is similar to the one above, the CPO will not know the difference.

Figure 37. Smart Charging Topology: The eMSP generates ChargingProfiles.

Connection	Interface	Role
SCSP - eMSP	Sender	SCSP
SCSP - eMSP	Receiver	eMSP
eMSP - CPO	Sender	eMSP
eMSP - CPO	Receiver	CPO

14.1.3. The CPO delegated Smart Charging to SCSP.

In this topology, the CPO has delegated the generation of ChargingProfiles to a SCSP. For this, the CPO and SCSP have agreed to use OCPI as the interface.

The CPO 'owns' the EVSE on which charging happens. As the CPO does not 'own' the customers, the CPO needs to

make sure the EV driver knows that the charging speed might not be the maximum the driver has expected, this could be something as simple as a sticker on the Charge Point, or might even be part of the tariff text.

The CPO might generate ChargingProfiles themselves, but as OCPI is then not used this is not part of this document.

The CPO can forward OCPI Session Objects to the SCSP. the SCSP can act on the received/updated Session Objects, by sending Charging Profile commands to the CPO.

The CPO and SCSP have to take into account that they have to oblige to local privacy laws when exchanging information about eMSPs customers.

In this topology, the eMSP is not aware that the CPO is using OCPI to receive Charging Profiles from the SCSP.

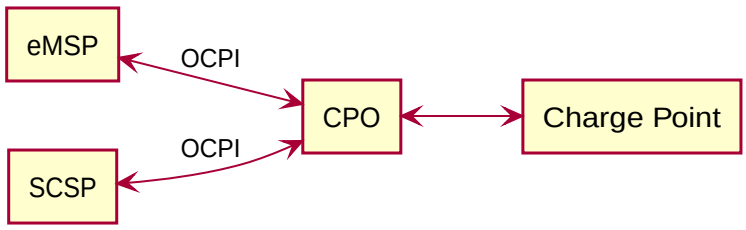


Figure 38. Smart Charging Topology: The eMSP generates ChargingProfiles.

Interface	Role
Sender	SCSP
Receiver	CPO

14.2. Use Cases

This module is designed to support the following use cases, for all the above mentioned topologies.

- The eMSP/SCSP sends/updates a ChargingProfile to manipulate an ongoing charging session.
- The eMSP/SCSP request to remove the set ChargingProfile from an ongoing charging session.
- The eMSP/SCSP request the ActiveChargingProfile for an ongoing charging session.
- The CPO updates the eMSP/SCSP of changes to an ActiveChargingProfile.

14.3. Flow

The ChargingProfile creation is a request to activate a charging profile on a running charging session.

Most Charge Points are hooked up to the internet via a relative slow wireless connection. To prevent long blocking calls, the ChargingProfile module is designed to work asynchronously. (similar to the `Commands` module.

The Sender (Typically SCSP) sends a request to a Receiver (Typically CPO), via the Receiver interface. The Receiver checks if it can send the request to a Charge Point and will respond to the request with a status, indicating if the request can be sent to a Charge Point.

The Receiver sends the requested command (via another protocol, for example: OCPP) to a Charge Point. The Charge Point will respond if it understands the command and will try to execute the command. This response doesn't

always mean that the ChargingProfile will be executed. The CPO will forward the result in a new POST request to the Sender (Typically SCSP) ChargingProfile interface.

The Sender (Typically SCSP) can send the Charging Profile to the EVSE via the CPO by using the CPO PUT method for an ongoing session. The Sender can request the current profile the EVSE has calculated, based on different inputs, and is planned to be used for the ongoing session by calling the CPO GET method. The Sender has the ability to remove the Charging Profile for the session by calling the CPO DELETE method

When the Sender has (at least once) successfully sent a Charging Profile for an ongoing charging session, the Receiver (Typically CPO) SHALL keep the Sender updated with changes to the ActiveChargingProfile of that Session. If the Receiver is aware of any changes, he notifies the Sender by calling the MSP PUT method. The changes might be triggered by the CPO sending additional Charging Profiles, or the some local limit being applied to the Charge Point, and the Charge Point notifies the CPO of the Changes.

The Receiver can cancel/remove an existing ChargingProfile, it can let the eMSP know by calling the MSP PUT method

For calculating optimum ChargingProfiles it might be useful for the eMSP or SCSP to know the ChargingProfile that the Charge Point has planned for the Session: ActiveChargingProfile. The ActiveChargingProfile might differ from ChargingProfile requested via OCPI. There might be other limiting factors being taken into account by the CPO and or Charge Point, that limit the ChargingProfile. The ActiveChargingProfile profile can be requested by the Sender by calling the CPO GET method on the Charging Profile Receiver interface. The CPO will then ask the Charge Point for the planned ActiveChargingProfile. When that is received it is forwarded to the URL given by the eMSP or SCSP.

The CPO can limit the amount of request that can be done on the Charging Profiles interface, this too prevent creating a too high load or data usages. To do this the CPO can reject a request on the Charging Profile Receiver interface be responding with: TOO_OFTEN.

If the Sender (typically eMSP or SCSP) wants to have a reference between the calls sent to the Receivers interface and the asynchronous result received from the Charge Point via the CPO, the Sender can make some unique identifier part of the `response_url` that is part of every method in the Receiver interface. The Receiver will call this URL when the result is received from the Charge Point. The Sender can then match the unique identifier from the URL called with the request.

14.3.1. Example of setting/updating a ChargingProfile by the Sender (typically the SCSP or eMSP)

When a new Session is started, or when an update to an existing Session is available, the CPO sends the Session object to the eMSP or SCSP. The eMSP or SCSP calculates a Charging Profile and sends it to the CPO by calling the Charging Profiles PUT method on the Receiver interface.

The CPO responds to the eMSP or SCSP, the response body will contain the response to the request, acknowledging the request was understood and can be forwarded to the Charge Point.

The CPO sends the requests to the Charge Point. When the CPO receives a response from the Charge Point, that result is sent to the eMSP or SCSP by call the POST method, on the URL provided by the eMSP of SCSP in the PUT request, this call will contain a ChargingProfileResult Object.

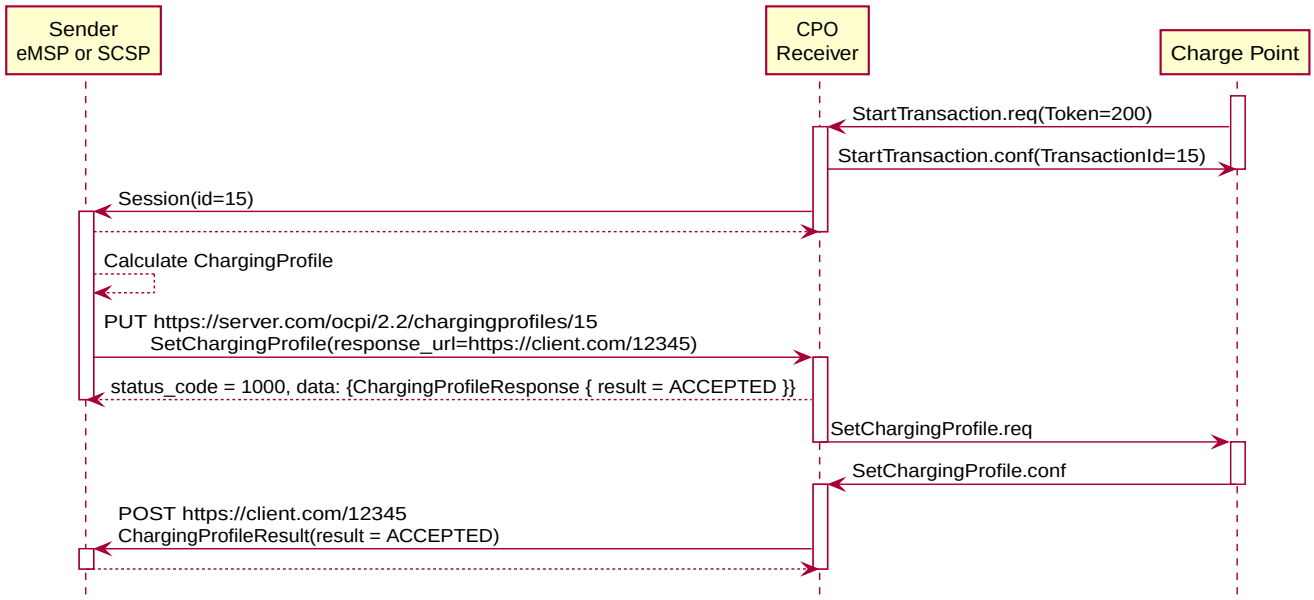


Figure 39. Example of a SetChargingProfile.

14.3.2. Example of a setting/updating a ChargingProfile by the SCSP via the eMSP

When a new Session is started, the CPO sends the Session object to the eMSP, the eMSP forwards the Session object to the SCSP.

When a new Session is started, or when an update to an existing Session is available, the CPO sends the Session object to the eMSP. The eMSP forwards the Session Object to the SCSP. The SCSP calculates a Charging Profile and sends it to the eMSP by calling the Charging Profiles PUT method on the Sender interface implemented by the eMSP. The eMSP forwards it to the CPO by calling the Charging Profiles PUT method on the Receiver interface.

The CPO responds to the eMSP, the response body will contain the response to the request, acknowledging the request was understood and can be forwarded to the Charge Point. The eMSP forwards this response to the SCSP.

The CPO sends the requests to the Charge Point. When the CPO receives a response from the Charge Point, that result is sent to the eMSP by the POST method, on the URL provided by the eMSP in the PUT request from the eMSP. The eMSP forwards this result to the the URL provided by the SCSP in the PUT request of the SCSP, this call will contain a ChargingProfileResult Object.

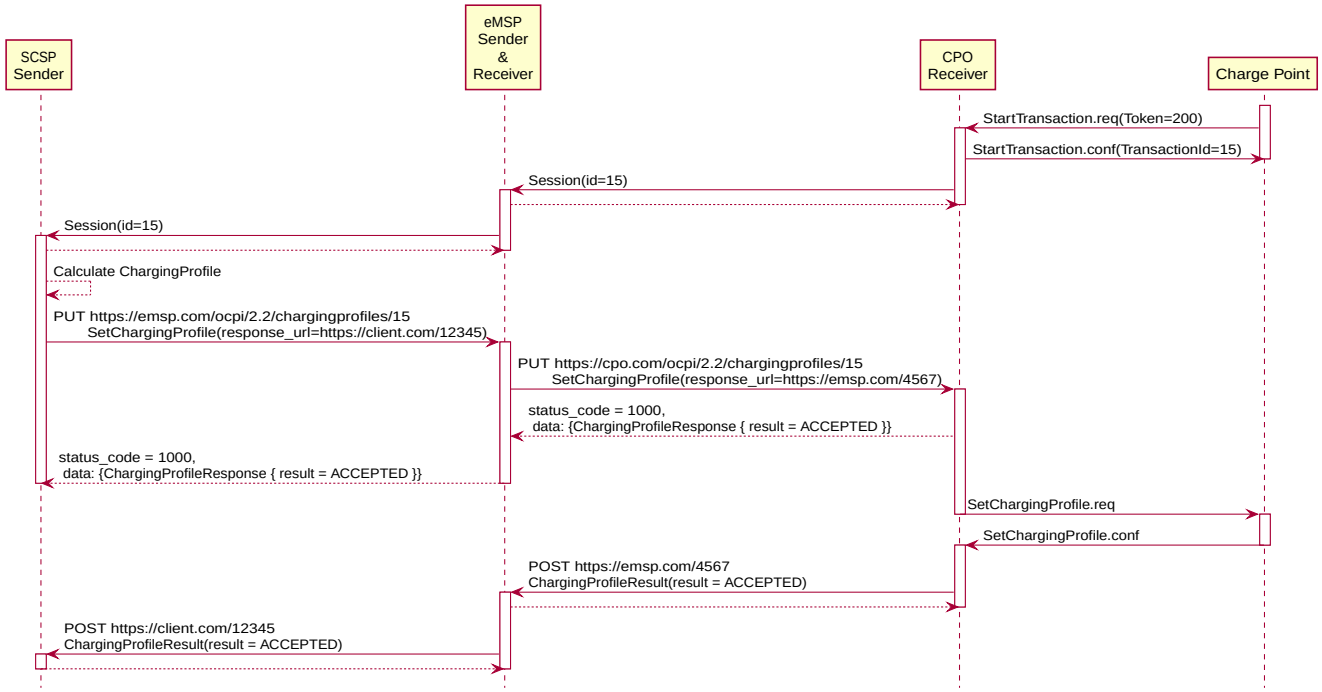


Figure 40. Example of a SetChargingProfile via the MSP.

14.3.3. Example of a removing/clearing ChargingProfile sent by the Sender (typically the eMSP or SCSP)

The Sender might want to remove the charging profile, for example the EV driver has selected to switch to charging with the highest speed possible. The Sender can ask the CPO to remove the set charging profile. This can be done by calling the DELETE method on the Receiver interface.

The CPO responds to the eMSP or SCSP, the response body will contain the response to the request, acknowledging the request was understood and can be forwarded to the Charge Point.

The CPO sends the clear requests to the Charge Point. When the CPO receives a response from the Charge Point, that result is sent to the eMSP by call the POST method, on the URL provided by the eMSP in the DELETE request of the eMSP, this call will contain a ClearProfileResult Object.

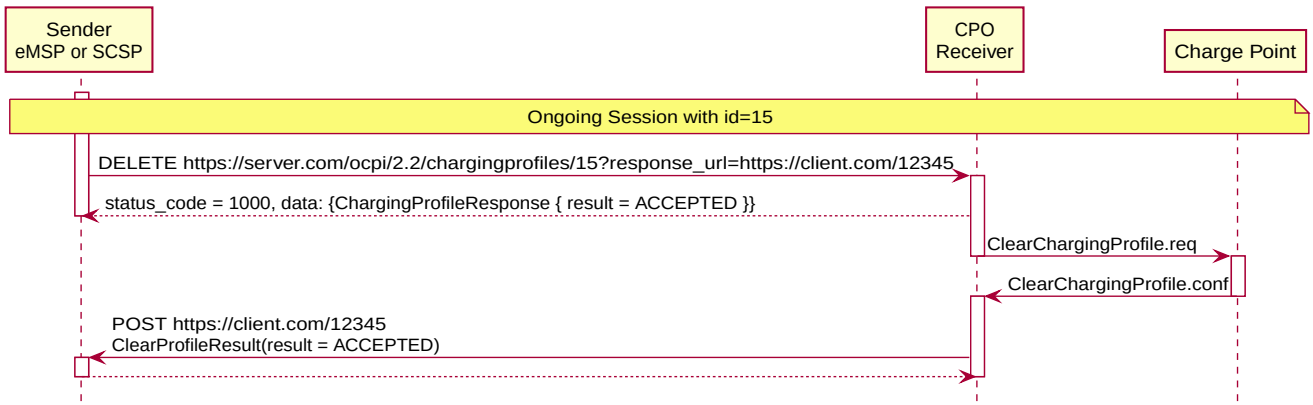


Figure 41. Example of a ClearChargingProfile.

14.3.4. Example of a removing/clearing ChargingProfile send by the SCSP via the eMSP

The SCSP might want to remove the charging profile, for example the EV driver has selected to switch to charging

with the highest speed possible. The SCSP can ask the eMSP to ask the CPO to remove the set charging profile. This can be done by calling the DELETE method on the eMSPs Charging Profile Receiver interface. The eMSP forwards this to the CPO by calling the DELETE method on the CPOs Charging Profile Receiver interface.

The CPO responds to the eMSP, the response body will contain the response to the request, acknowledging the request was understood and can be forwarded to the Charge Point. The eMSP forwards this response to the SCSP.

The CPO send the clear requests to the Charge Point. When the CPO receives a response from the Charge Point, that result is sent to the eMSP by call the POST method, on the URL provided by the eMSP in the DELETE request of the eMSP. The eMSP forwards this result to the the URL provided by the SCSP in the DELETE request of the SCSP, this call will contain a ClearProfileResult Object.

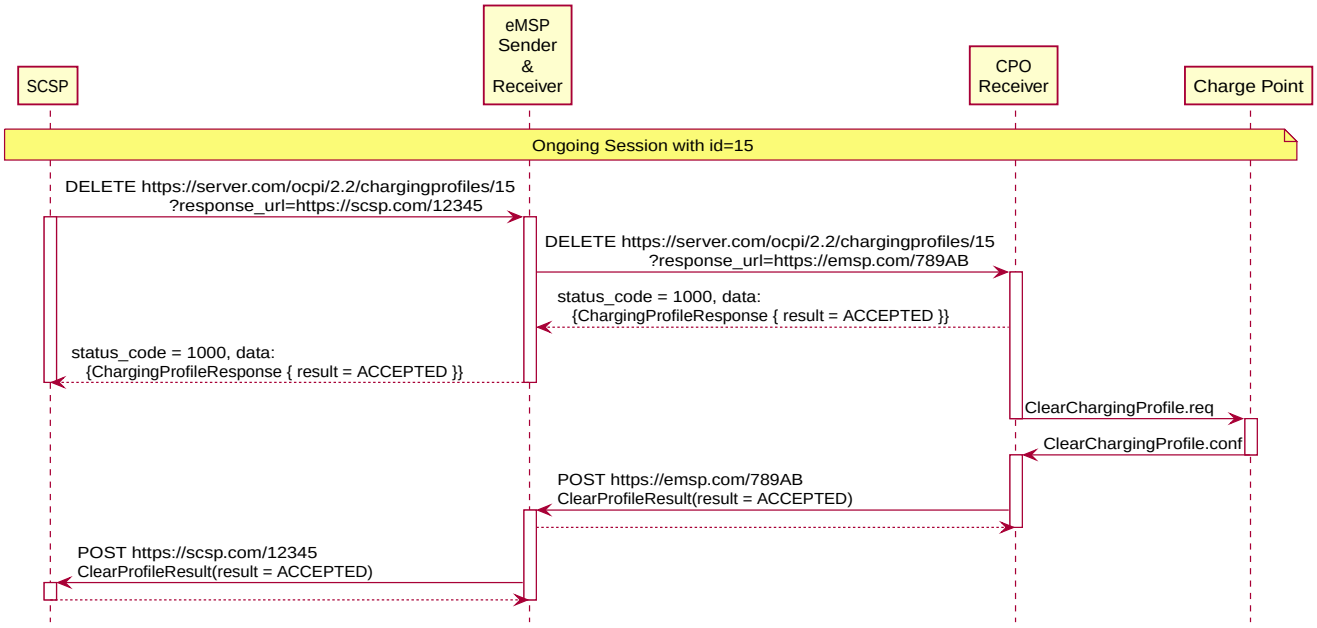


Figure 42. Example of a ClearChargingProfile via the MSP.

14.3.5. Example of a GET ActiveChargingProfile send by the Sender (typically the eMSP or SCSP)

When the Sender wants to know the current planned charging profile for a session, the Sender can ask the CPO for the ActiveChargingProfile by calling the GET method on the Receiver interface.

The CPO responds to the eMSP or SCSP, the response body will contain the response to the request, acknowledging the request was accepted and can be forwarded to the Charge Point.

The CPO sends a message to the Charge Point to retrieve the current active charging profile. When the CPO receives a response from the Charge Point, that ActiveChargingProfile is sent to the eMSP by call the POST method, on the URL provided by the eMSP in the GET request of the eMSP, this call will contain a ActiveChargingProfileResult Object.

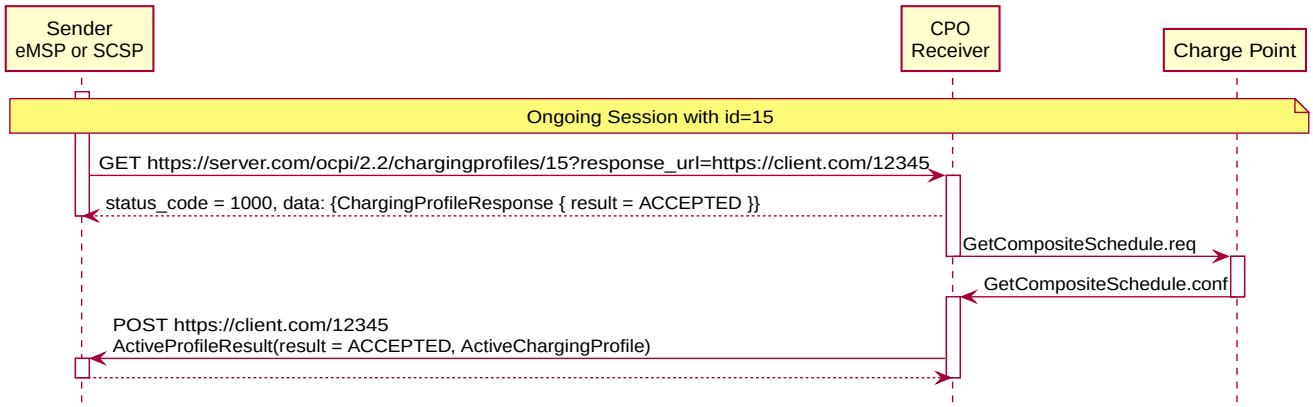


Figure 43. Example of a GET ActiveChargingProfile.

14.3.6. Example of a GET ActiveChargingProfile send by the SCSP via eMSP

When the SCSP wants to known the current planned charging profile for a session, the SCSP can ask the the eMSP to ask the CPO for the ActiveChargingProfile by calling the GET method on the eMSPs Charging Profile Receiver interface. The eMSP forwards this to the CPO by calling the GET method on the CPOs Charging Profile Receiver interface.

The CPO responds to the eMSP, the response body will contain the response to the request, acknowledging the request was accepted and can be forwarded to the Charge Point. The eMSP forwards this response to the SCSP.

The CPO sends a message to the Charge Point to retrieve the current active charging profile. When the CPO receives a response from the Charge Point, that ActiveChargingProfile is sent to the eMSP by call the POST method, on the URL provided by the eMSP in the GET request of the eMSP, this call will contain a ActiveChargingProfileResult Object. The eMSP forwards this result to the the URL provided by the SCSP in the GET request of the SCSP, this call will contain the same ActiveChargingProfileResult Object.

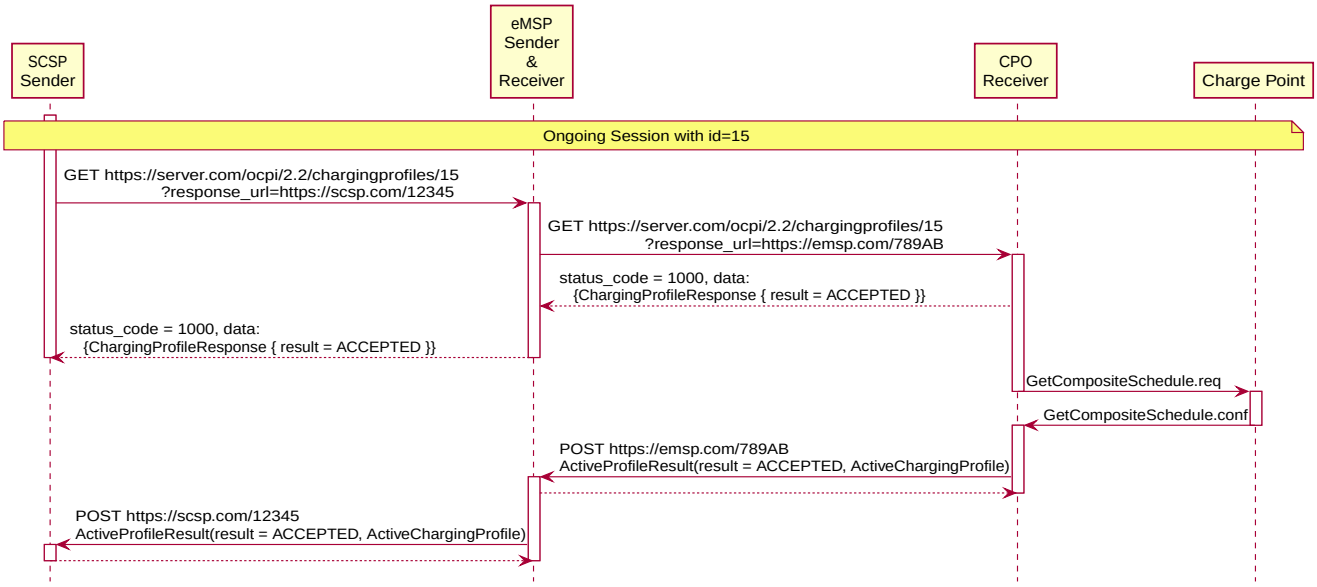


Figure 44. Example of a GET ActiveChargingProfile via the MSP.

14.3.7. Example of the Receiver (typically the CPO) sending an updated ActiveChargingProfile

When the CPO knows the ActiveChargingProfile of a Charge Point has changed, the Receiver (typically the CPO) sends this update ActiveChargingProfile to the Sender (typically the eMSP or SCSP), by calling the PUT method on the

Sender interface.

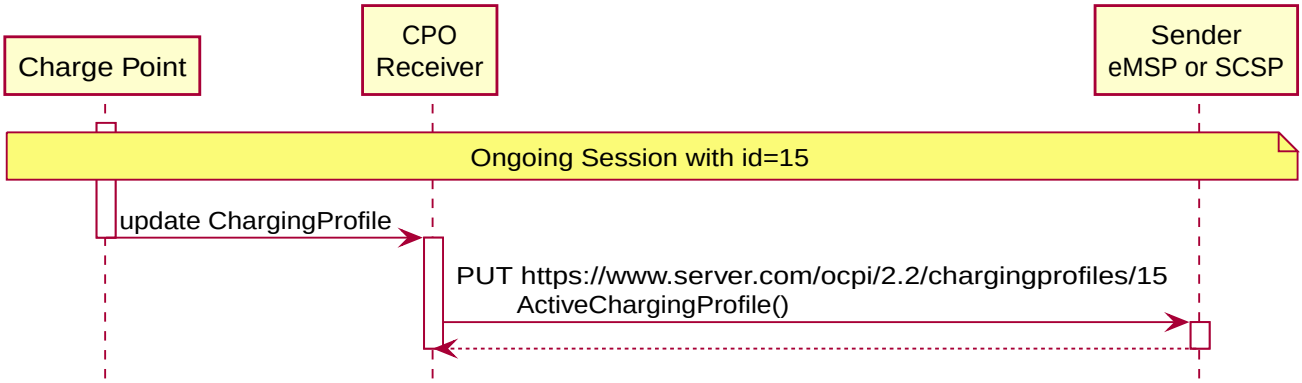


Figure 45. Example of an ActiveChargingProfile being send by the CPO

14.3.8. Example of the Receiver (typically the CPO) sending an updated ActiveChargingProfile to the SCSP via the eMSP

When the CPO knows the ActiveChargingProfile of a Charge Point has changed, the Receiver (typically the CPO) sends this update ActiveChargingProfile to the Sender (SCSP), by calling the PUT method on the eMSPs Sender interface.

The eMSP forwards this ActiveChargingProfile to the SCSP, by calling the PUT method on the SCSPs Sender interface.

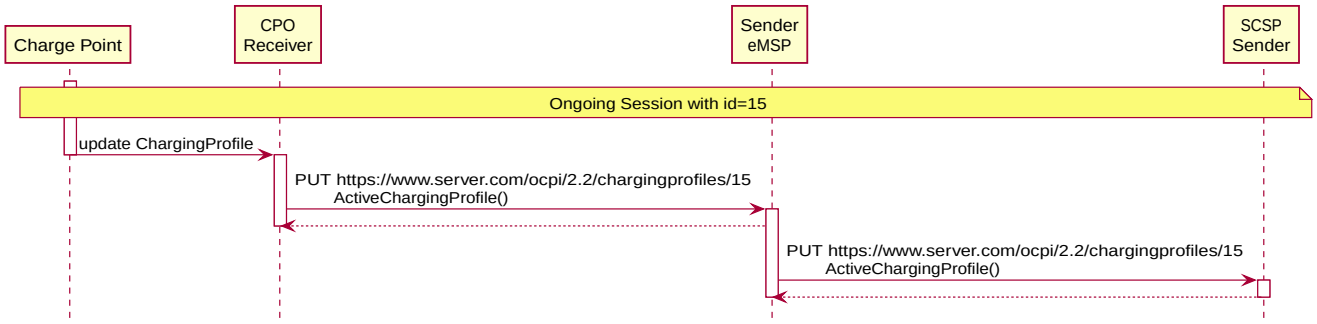


Figure 46. Example of an ActiveChargingProfile being sent by the CPO via the eMSP

14.4. Interfaces and endpoints

The ChargingProfiles module consists of two interfaces: a Receiver interface that enables a Sender (and its clients) to send ChargingProfiles to a Location/EVSE, and an Sender interface to receive the response from the Location/EVSE asynchronously.

14.4.1. Receiver Interface

Typically implemented by market roles like: CPO.

Example endpoint structures:

Method	Description
GET	Gets the ActiveChargingProfile for a specific charging session.
POST	n/a

Method	Description
PUT	Creates/updates a ChargingProfile for a specific charging session.
PATCH	n/a
DELETE	Cancels an existing ChargingProfile for a specific charging session.

14.4.1.1. GET Method

Retrieves the ActiveChargingProfile as it is currently planned for the the given session.

Endpoint structure definition:

`{chargingprofiles_endpoint_url}{session_id}?duration={duration}&response_url={url}`

Example:

`https://www.cpo.com/ocpi/2.2.1/chargingprofiles/1234?duration=900&response_url=https://www.msp.com/ocpi/2.2.1/chargingprofile/response?request_id=5678`

NOTE As it is not common to add a body to a GET request, all parameters are added to the URL.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the Receiver platform.
duration	int	yes	Length of the requested ActiveChargingProfile in seconds Duration in seconds. *
response_url	URL	yes	URL that the ActiveChargingProfileResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between GET ActiveChargingProfile requests.

NOTE duration: Balance the duration between maximizing the information gained and the data usage and computation to execute on the request. Warning: asking for longer duration than necessary might result in additional data costs, while its added value diminishes with every change in the schedule.

Response Data

The response contains the direct response from the Receiver, not the response from the EVSE itself. That information will be sent via an asynchronous POST on the Sender interface if this response is `ACCEPTED`.

Datatype	Card.	Description
ChargingProfileResponse	1	Result of the ActiveChargingProfile request, by the Receiver (Typically CPO), not the location/EVSE. So this indicates if the Receiver understood the ChargingProfile request and was able to send it to the EVSE. This is not the response by the Charge Point.

14.4.1.2. PUT Method

Creates a new ChargingProfile on a session, or replaces an existing ChargingProfile on the EVSE.

Endpoint structure definition:

`{chargingprofiles_endpoint_url}{session_id}`

Example:

`https://www.cpo.com/ocpi/2.2.1/chargingprofiles/1234`

Request Parameters

The following parameter shall be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the Receiver platform.

14.4.1.3. Request Body

The body contains a SetChargingProfile object, that contains the new ChargingProfile and a response URL.

Type	Card.	Description
SetChargingProfile	1	SetChargingProfile object with information needed to set/update the Charging Profile for a session.

Response Data

The response contains the direct response from the Receiver (Typically CPO), not the response from the EVSE itself, that will be sent via an asynchronous POST on the Sender interface if this response is `ACCEPTED`.

Datatype	Card.	Description
ChargingProfileResponse	1	Result of the ChargingProfile PUT request, by the CPO (not the location/EVSE). So this indicates if the CPO understood the ChargingProfile PUT request and was able to send it to the EVSE. This is not the response by the Charge Point.

14.4.1.4. DELETE Method

Clears the ChargingProfile set by the eMSP on the given session.

Endpoint structure definition:

`{chargingprofiles_endpoint_url}{session_id}?response_url={url}`

Example:

`https://www.cpo.com/ocpi/2.2.1/chargingprofiles/1234?response_url=https://www.server.com/example`

NOTE	As it is not common to add a body to a DELETE request, all parameters are added to the URL.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the Receiver platform.
response_url	URL	yes	URL that the ClearProfileResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between DELETE ChargingProfile requests.

Response Data

The response contains the direct response from the Receiver (typically CPO), not the response from the EVSE itself, that will be sent via an asynchronous POST on the Sender interface if this response is `ACCEPTED`.

Datatype	Card.	Description
ChargingProfileResponse	1	Result of the ChargingProfile DELETE request, by the CPO (not the location/EVSE). So this indicates if the CPO understood the ChargingProfile DELETE request and was able to send it to the EVSE. This is not the response by the Charge Point.

14.4.2. Sender Interface

Typically implemented by market roles like: SCSP.

The Sender interface receives the asynchronous responses.

Method	Description
GET	n/a
POST	Receive the asynchronous response from the Charge Point.

Method	Description
PUT	Receiver (typically CPO) can send an updated ActiveChargingProfile when other inputs have made changes to existing profile. When the Receiver (typically CPO) sends a update profile to the EVSE, for an other reason then the Sender (Typically SCSP) asking, the Sender SHALL post an update to this interface. When a local input influence the ActiveChargingProfile in the EVSE AND the Receiver (typically CPO) is made aware of this, the Receiver SHALL post an update to this interface.
PUT	n/a
PATCH	n/a
DELETE	n/a

14.4.2.1. POST Method

Request Parameters

There are no URL segment parameters required by OCPI.

As the Sender interface is called by the Receiver (typically CPO) on the URL given `response_url` in the Sender request to the Receiver interface. It is up to the implementation of the Sender (typically SCSP) to determine what parameters are put in the URL. The Sender sends a URL in the POST method body to the Receiver. The Receiver is required to use this URL for the asynchronous response by the Charge Point. It is advised to make this URL unique for every request to differentiate simultaneous commands, for example by adding a unique id as a URL segment.

Endpoint structure definition:

No structure defined. This is open to the eMSP to define, the URL is provided to the Receiver by the Sender. Therefor OCPI does not define variables.

Examples:

`https://www.server.com/ocpi/2.2.1/chargingprofiles/chargingprofile/12345678`

`https://www.server.com/activechargingprofile/12345678`

`https://www.server.com/clearprofile?request_id=12345678`

`https://www.server.com/ocpi/2.2.1/12345678`

The content of the request body depends on the original request by the eMSP to which this POST is send as a result.

14.4.2.2. Request Body

Datatype	Card.	Description
Choice: one of three		
ActiveChargingProfileResult	1	Result of the GET ActiveChargingProfile request, from the Charge Point.
ChargingProfileResult	1	Result of the PUT ChargingProfile request, from the Charge Point.

Datatype	Card.	Description
ClearProfileResult	1	Result of the DELETE ChargingProfile request, from the Charge Point.

14.4.2.3. Response Body

The response to the POST on the Sender interface SHALL contain the Response Format with the data field omitted.

14.4.2.4. PUT Method

Updates the Sender (typically SCSP) when the Receiver (typically CPO) knows the ActiveChargingProfile has changed.

The Receiver SHALL call this interface every time it knows changes have been made that influence the ActiveChargingProfile for an ongoing session AND the Sender has at least once successfully called the charging profile Receiver PUT interface for this session (SetChargingProfile). If the Receiver doesn't know the ActiveChargingProfile has changed (EVSE does not notify the Receiver (typically CPO) of the change) it is not required to call this interface.

The Receiver SHALL NOT call this interface for any session where the Sender has never, successfully called the charging profile Receiver PUT interface for this session (SetChargingProfile).

The Receiver SHALL send a useful relevant duration of ActiveChargingProfile to send to the Sender. As a guide: between 5 and 60 minutes. If the Sender wants a longer ActiveChargingProfile the Sender can always do a GET with a longer duration.

Endpoint structure definition:

```
{chargingprofiles_endpoint_url}{session_id}
```

Example:

```
https://www.server.com/ocpi/2.2.1/chargingprofiles/1234
```

Request Parameters

The following parameter shall be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the Receiver platform.

14.4.2.5. Request Body

The body contains the update ActiveChargingProfile, The ActiveChargingProfile is the charging profile as calculated by the EVSE.

Type	Card.	Description
ActiveChargingProfile	1	The new ActiveChargingProfile. If there is no longer any charging profile active, the ActiveChargingProfile SHALL reflect this by showing the maximum charging capacity of the EVSE.

14.4.2.6. Response Body

The response to the PUT on the eMSP interface SHALL contain the Response Format with the data field omitted.

14.5. Object description

14.5.1. *ChargingProfileResponse* Object

The ChargingProfileResponse object is send in the HTTP response body.

Because OCPI does not allow/require retries, it could happen that the asynchronous result url given by the eMSP is never successfully called. The eMSP might have had a glitch, HTTP 500 returned, was offline for a moment etc. For the eMSP to be able to reject to timeouts, it is important for the eMSP to know the timeout on a certain command.

Property	Type	Card.	Description
result	ChargingProfileResponseType	1	Response from the CPO on the ChargingProfile request.
timeout	int	1	Timeout for this ChargingProfile request in seconds. When the Result is not received within this timeout, the eMSP can assume that the message might never be sent.

14.5.2. *ActiveChargingProfileResult* Object

The ActiveChargingProfileResult object is send by the CPO to the given `response_url` in a POST request. It contains the result of the GET (ActiveChargingProfile) request send by the eMSP.

Property	Type	Card.	Description
result	ChargingProfileResultType	1	The EVSE will indicate if it was able to process the request for the ActiveChargingProfile
profile	ActiveChargingProfile	?	The requested ActiveChargingProfile, if the result field is set to: `ACCEPTED`

14.5.3. *ChargingProfileResult* Object

The ChargingProfileResult object is send by the CPO to the given `response_url` in a POST request. It contains the result of the PUT (SetChargingProfile) request send by the eMSP.

Property	Type	Card.	Description
result	ChargingProfileResultType	1	The EVSE will indicate if it was able to process the new/updated charging profile.

14.5.4. *ClearProfileResult* Object

The ClearProfileResult object is send by the CPO to the given `response_url` in a POST request. It contains the result of the DELETE (ClearProfile) request send by the eMSP.

Property	Type	Card.	Description
result	ChargingProfileResultType	1	The EVSE will indicate if it was able to process the removal of the charging profile (ClearChargingProfile).

14.5.5. *SetChargingProfile* Object

Object set to a CPO to set a Charging Profile.

Property	Type	Card.	Description
charging_profile	ChargingProfile	1	Contains limits for the available power or current over time.
response_url	URL	1	URL that the ChargingProfileResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between GET ActiveChargingProfile requests.

14.6. Data types

14.6.1. ActiveChargingProfile *class*

Property	Type	Card.	Description
start_date_time	DateTime	1	Date and time at which the Charge Point has calculated this ActiveChargingProfile. All time measurements within the profile are relative to this timestamp.
charging_profile	ChargingProfile	1	Charging profile structure defines a list of charging periods.

14.6.2. ChargingRateUnit *enum*

Unit in which a charging profile is defined.

Value	Description
W	Watts (power) This is the total allowed charging power. It is usually more convenient to use this for DC charging.
A	Amperes (current) The amount of Ampere per phase, not the sum of all phases. It is usually more convenient to use this for AC charging.

14.6.3. ChargingProfile *class*

Charging profile class defines a list of charging periods.

Property	Type	Card.	Description
start_date_time	DateTime	?	Starting point of an absolute profile. If absent the profile will be relative to start of charging.
duration	int	?	Duration of the charging profile in seconds. If the duration is left empty, the last period will continue indefinitely or until end of the transaction in case start_date_time is absent.
charging_rate_unit	ChargingRateUnit	1	The unit of measure.
min_charging_rate	number	?	Minimum charging rate supported by the EV. The unit of measure is defined by the chargingRateUnit. This parameter is intended to be used by a local smart charging algorithm to optimize the power allocation for in the case a charging process is inefficient at lower charging rates. Accepts at most one digit fraction (e.g. 8.1)
charging_profile_period	ChargingProfilePeriod	*	List of ChargingProfilePeriod elements defining maximum power or current usage over time.

14.6.4. ChargingProfilePeriod *class*

Charging profile period structure defines a time period in a charging profile, as used in: ChargingProfile

Property	Type	Card.	Description
start_period	int	1	Start of the period, in seconds from the start of profile. The value of StartPeriod also defines the stop time of the previous period.
limit	number	1	Charging rate limit during the profile period, in the applicable chargingRateUnit, for example in Amperes (A) or Watts (W). Accepts at most one digit fraction (e.g. 8.1).

14.6.5. ChargingProfileResponseType *enum*

Response to the ChargingProfile request from the eMSP to the CPO.

Value	Description
ACCEPTED	ChargingProfile request accepted by the CPO, request will be forwarded to the EVSE.
NOT_SUPPORTED	The ChargingProfiles not supported by this CPO, Charge Point, EVSE etc.
REJECTED	ChargingProfile request rejected by the CPO. (Session might not be from a customer of the eMSP that send this request)
TOO_OFTEN	ChargingProfile request rejected by the CPO, requests are send more often then allowed.
UNKNOWN_SESSION	The Session in the requested command is not known by this CPO.

14.6.6. ChargingProfileResultType *enum*

Result of a ChargingProfile request that the EVSE sends via the CPO to the eMSP.

Value	Description
ACCEPTED	ChargingProfile request accepted by the EVSE.
REJECTED	ChargingProfile request rejected by the EVSE.
UNKNOWN	No Charging Profile(s) were found by the EVSE matching the request.

15. *HubClientInfo* module

Module Identifier: `hubclientinfo`

Data owner: `Hub`

Type: Configuration Module

This module provides parties connected to a hub with the connection status of other parties that are connected to a hub that they can communicate with. So, CPOs know which eMSP and other parties are online and vice versa.

Unlike the usual OCPI modules, this module is between eMSP/CPO and Hub instead of between eMSP and CPO.

15.1. Scenarios

This section will describe what the expected behavior is when a party receives information of a ConnectionState change.

15.1.1. Another Party becomes CONNECTED

Party is (back) online. Request can be sent again. Every party receiving Client Owned Objects from this party should be prepared to receive Client Owned Objects with URLs that contain the party_id and country_code of this party.

15.1.2. Another Party goes OFFLINE

Connection to party is not available: No requests can be sent. Do not queue Push messages. When the other party comes back online, it is their responsibility to do a GET to get back in sync.

15.1.3. Another Party becomes PLANNED

No requests can be sent to this new party yet. It can be a good idea to sent some notification to an operator to get into contact with the new party so contracts can be setup. This state may also be used when a Hub has some configuration indicating which parties have contracts which each other. When a company does not have a connection configured, this state may also be sent to parties.

15.1.4. Another Party becomes SUSPENDED

Like with OFFLINE, no requests should be sent to this party, they cannot be delivered.

When, for example, CDRs still have to be delivered (there is some unfinished business) parties are advised to get into contact with the other party in some other way: call them, or send an e-mail.

15.2. Flow and Life-cycle

15.2.1. Push model

When the Hub creates a new ClientInfo object they push it to the connected parties by calling PUT on the connected party ClientInfo endpoint with the newly created ClientInfo object.

Any changes to ClientInfo in the Hub system are sent to the connected party system by calling the PUT method on the connected party ClientInfo endpoint with the updated ClientInfo.

When the Hub invalidates a ClientInfo object (deleting is not possible), the Hub will send the updated ClientInfo object (with the field: status set to SUSPENDED, by calling the PUT method on the connected party ClientInfo endpoint with the updated ClientInfo object.

When the connected party is not sure about the state or existence of a ClientInfo object in the Hub system, the connected party can call the GET to request to ClientInfo object from the Hub system.

15.2.2. Pull model

When a connected party is not sure about the state of the list of known connected parties of a Hub, or wants to request the full list at the start-up of their system, the connected party can call the GET on the Hubs ClientInfo endpoint to receive all ClientInfo objects. This method is not for operational flow.

15.2.3. Still alive check.

The hubs needs to determine if a connection is still "alive".

To do this, the Hub should keep track of the time that has passed since the last message was received from a connected party. When this is longer then X minutes (when unsure, start with 5 minutes) the Hub should send a: GET to the Version information endpoint. As the Version information endpoint is always required in OCPI, and this endpoint is provided by all parties, and a GET to the versions endpoint does not have any side effects, this is seen as the best way to do an "still-alive"check.

15.3. Interfaces

There is both a Sender (Typically Hub) as a Receiver interface for ClientInfo. It is advised to use the Push direction from Sender to connected clients during normal operation. The Hub interface is meant to be used when the connected client is not 100% sure the ClientInfo cache is still correct.

15.3.1. Receiver Interface

Typically implemented by all parties connecting to a Hub.

With this interface the Hub can push the ClientInfo information to a connected client (eMSP/CPO etc) Example endpoint structure: `/ocpi/cpo/2.0/clientinfo/{country_code}/{party_id}`

Method	Description
GET	Retrieve a ClientInfo object as it is stored in the connected clients system.
POST	n/a
PUT	Push new/updated ClientInfo object to the connect client.
PATCH	n/a
DELETE	n/a, Use PUT, ClientInfo objects cannot be removed).

15.3.1.1. GET Method

If the Hub wants to check the status of a ClientInfo object in the connected clients system it might GET the object from the connected clients system for validation purposes. The Hub is the owner of the objects, so it would be illogical if the connected client system had a different status or was missing an object.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the requested ClientInfo object.
party_id	CiString(3)	yes	Party ID (Provider ID) of the requested ClientInfo object.

Response Data

The response contains the requested object.

Type	Card.	Description
ClientInfo	1	The requested ClientInfo object.

15.3.1.2. PUT Method

New or updated ClientInfo objects are pushed from the Hub to a connected client.

Request Body

In the put request a the new or updated ClientInfo object is send.

Type	Card.	Description
ClientInfo	1	New or updated ClientInfo object.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the eMSP sending this PUT request to the CPO system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the eMSP sending this PUT request to the CPO system.

Example: put a new ClientInfo object

```
PUT To URL: https://www.server.com/ocpi/cpo/2.0/clientinfo/NL/ALL

{
  "country_code": "NL",
  "party_id": "ALL",
  "role": "CPO",
  "status": "PLANNED",
}
```

15.3.2. Sender Interface

Typically implemented by the Hub.

This interface enables Receivers to request the current list of ClientInfo objects from the Sender, when needed.

Method	Description
GET	Get the list of known ClientInfo objects, last updated between the {date_from} and {date_to} paginated)
POST	n/a
PUT	n/a
PATCH	n/a
DELETE	n/a

15.3.2.1. GET Method

Fetch information about clients connected to a Hub.

Endpoint structure definition:

`{locations_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/cpo/2.2.1/hubclientinfo/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/hubclientinfo/?offset=50`

`https://www.server.com/ocpi/2.2.1/hubclientinfo/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/cpo/2.2.1/hubclientinfo/?offset=50&limit=100`

15.3.2.2. Request Parameters

If additional parameters: `{date_from}` and/or `{date_to}` are provided, only ClientInfo objects with (`last_updated`) between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned.

This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	no	Only return ClientInfo that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	no	Only return ClientInfo that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

15.3.2.3. Response Data

The endpoint response with list of valid ClientInfo objects, the header will contain the pagination related headers.

Any older information that is not specified in the response is considered as no longer valid. Each object must contain all required fields. Fields that are not specified may be considered as null values.

Type	Card.	Description
ClientInfo	*	List of all (or matching) ClientInfo objects.

15.4. Object description

15.4.1. *ClientInfo* Object

Property	Type	Card.	Description
party_id	CiString(3)	1	CPO or eMSP ID of this party (following the 15118 ISO standard), as used in the credentials exchange.
country_code	CiString(2)	1	Country code of the country this party is operating in, as used in the credentials exchange.
role	Role	1	The role of the connected party.
status	ConnectionStatus	1	Status of the connection to the party.
last_updated	DateTime	1	Timestamp when this ClientInfo object was last updated.

15.5. Data types

15.5.1. ConnectionStatus *enum*

Value	Description
CONNECTED	Party is connected.
OFFLINE	Party is currently not connected.

Value	Description
PLANNED	Connection to this party is planned, but has never been connected.
SUSPENDED	Party is now longer active, will never connect anymore.

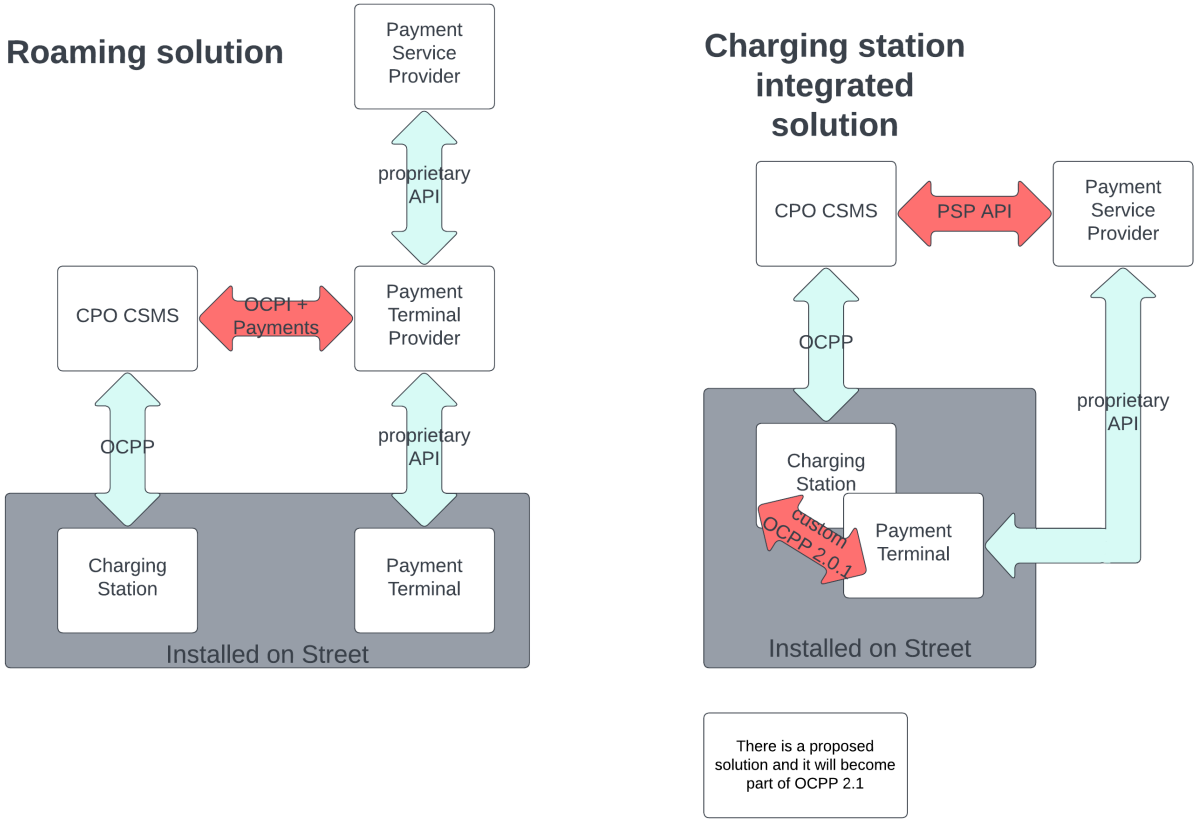
16. *Payments* module

Module Identifier: `payments`

Data owner: `PTP`

Type: Functional Module

This module should support the Payment Terminal use case for direct payment in the roaming world.



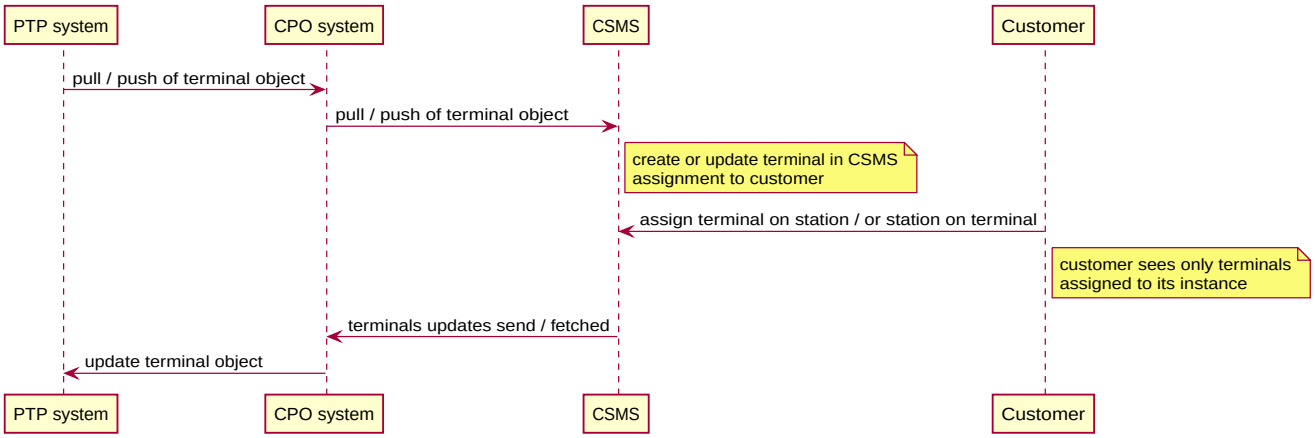
This should show the difference between roaming and an integrated charging station solution for direct payment support.

The module consists mainly of two objects: Terminal and Financial Advice Confirmation. A terminal can handle multiple locations and/or EVSEs. It should be able for a CPO to assign them to a terminal object. At the end of a charging session there should be a CDR sent. There should be also send a financial-advice-confirmation from the PTP. This should contain the actual cost and EFT data. This object is only needed if the CPO creates the invoice.

16.1. Usage Flows

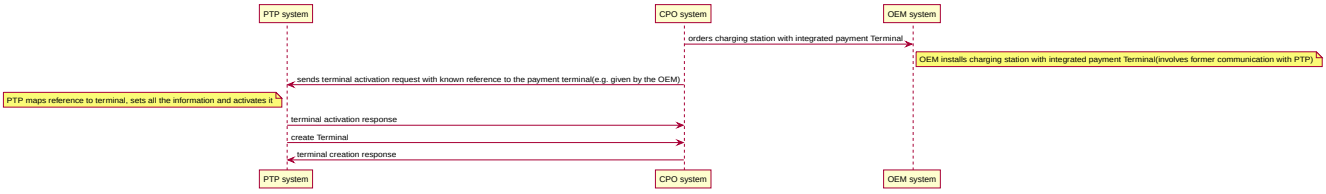
16.2. Terminal Assignment

This flow shows the exchange and the assignment of the terminal object. This object will be owned and created by the PTP. After the object was pushed to/pulled by the CPO there will be the possibility to assign specific locations to this terminal. This assignment then will be pushed by the CPO to the PTP.



16.3. Terminal Activation

This flow shows a possible former activation of a payment terminal. Usually, this will be needed for payment terminals integrated into a station. Here the CPO orders a station from the OEM with an integrated payment terminal. The OEM will provide a reference to the CPO which can then be used for the terminal activation at the PTP. After the activation, the PTP will create a terminal object on the CPO side. This activation is needed as the PTP has to do several configuration steps in beforehand, like acquiring a unique ID for the given installation address.

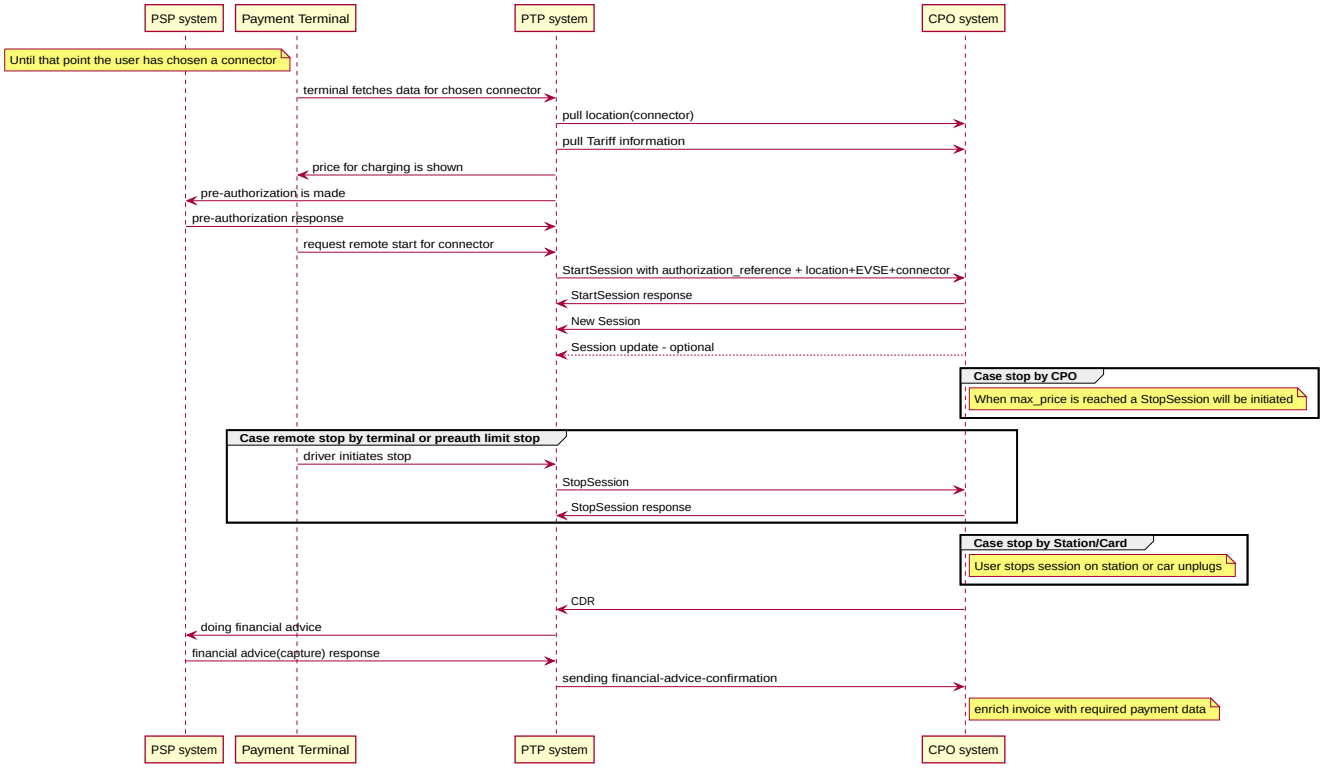


16.4. Transaction

This flow shows a single transaction in total. After choosing a specific connector on the terminal the PTP has to pull the current connector information to receive the applicable tariff_id. With this id the tariff should be pulled so that it can be shown on the terminal. Also the tariff is needed to reserve the needed preauth limit (stated in the `preauthorize_amount` field of the tariff) at the PSP. When the user accepts the tariff the PTP will reserve the preauth amount at the PTP and a StartSession command will be sent to the CPO. The PTP will pass an authorization_reference with this request which will be used as the mapping for the invoice (e.g. as postfix of the invoice URL). During the session there will be session updates pushed by the CPO if existing. For stopping the session there are 3 ways:

1. can be stopped by the car or by the station
2. a StopSession can be sent by the PTP when the preauth limit is reached
3. can also be stopped by the CPO backend if the preauth limit is reached

Now the PTP will receive a CDR with an filled invoice_reference_id. If this id is set then the invoice was created by the CPO, if not then the invoice will be created by the PTP. If the CPO creates the invoice, the PTP has to push a financial-advice-confirmation object after he has done the capture at the PSP. When the CPO has received this object the previously created invoice has to be enriched with the required EFT data.



16.5. Interfaces and Endpoints

16.5.1. Sender Interface

Typically implemented by market roles like: PTP.

16.5.1.1. Terminals Interface

Method	Description
GET	Fetch Terminal objects last updated between the {date_from} and {date_to}(paginated).
GET	Fetch a Terminal object by its ID.
POST	Activate a Terminal.
POST	Deactivate a Terminal.
PUT	Updating a Terminal object.
PATCH	Updating a Terminal object(Location assignment)

16.5.1.2. Financial Advice Confirmation Interface

Method	Description
GET	Fetch Financial Advice Confirmation objects last updated between the {date_from} and {date_to} (paginated).
GET	Fetch a Financial Advice Confirmation object by its ID.

16.5.1.3. GET Terminals Method

Fetch Terminals from a PTP system.

Endpoint structure definition:

`{payments_terminals_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/ptp/2.2.1/payments/terminals/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/payments/terminals/?offset=50`

`https://www.server.com/ocpi/2.2.1/payments/terminals/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/ptp/2.2.1/payments/terminals/?offset=50&limit=100`

Request Parameters

If the optional parameters date from and/or date to are provided, only Terminals with `last_update` between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned.

This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	no	Only return Sessions that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	no	Only return Sessions that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

Response Data

The response contains a list of Terminals objects that match the given parameters in the request, the header will contain the pagination related headers.

Any older information that is not specified in the response is considered no longer valid. Each object must contain all required fields. Fields that are not specified may be considered as null values.

Datatype	Card.	Description
Terminal	*	List of Terminal objects that match the request parameters.

16.5.1.4. GET Terminal Method

If the CPO wants to check the status of a Terminal object in the PTP system, it might GET the object from the PTP

system for validation purposes.

Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
terminal_id	CiString(36)	yes	Terminal.terminal id of the Terminal object to retrieve.

Response Data

The response contains the requested object.

Type	Card.	Description
> Terminal	1	Requested Terminal.

16.5.1.5. PATCH Terminal Method

This PATCH should be used by the CPO to assign location ids and/or evse_uids to a terminal. When sending both location_ids and evse_uids then both have to be considered and the sum of evses will be enabled for this payment terminal.

Request Parameters

This is an information Push message, the objects pushed will not be owned by the CPO.

Parameter	Datatype	Required	Description
terminal_id	CiString(36)	yes	Terminal.terminal id of the Terminal object to update.

Example: Assign Location IDs to Terminal

This is the expected type of update message. It is used to assign Location ids to a terminal.

```
PATCH To URL: https://www.server.com/ocpi/ptp/2.2.1/payments/terminals/55719888-
ed09-4cca-82cc-803bdb77bf26

{
  "location_ids": [
    "df37373d-1669-4127-a6ac-d86750095119",
    "a06dc823-3e5a-40c8-89cf-1b5b9e941412",
    "55719888-ed09-4cca-82cc-803bdb77bf26"
  ]
}
```

16.5.1.6. PUT Terminal Method

This PUT should be used by the CPO to update location data of a terminal.

Request Parameters

This is an information Push message, the objects pushed will not be owned by the CPO.

Parameter	Datatype	Required	Description
terminal_id	CiString(36)	yes	Terminal.terminal id of the Terminal object to update.

Request Body

The request body contains the updated object.

Type	Card.	Description
> Terminal	1	Terminal object to update.

Example: Setting customer_reference and invoice_base_url

```
PUT To URL: https://www.server.com/ocpi/ptp/2.2.1/payments/terminals/55719888-
ed09-4cca-82cc-803bdb77bf26

{
  "customer_reference": "OMV",
  "invoice_base_url": "someNewURL",
  "last_updated": "2021-01-01T00:00:00Z"
}
```

16.5.1.7. POST Activate Terminal Method

This POST request should be used by the CPO to activate a terminal with needed information on the PTP side. Activation of a terminal may be needed for payment terminals integrated into a station. Here the CPO has to give the PTP needed information to link the payment terminal to the location/evse. This can be done for example via the serial number or other mappings sent via the reference. After receiving an activation request the PTP should start the Terminal creation process by creating a Terminal object on the CPO side with calling the corresponding POST endpoint.

Endpoint structure definition:

`{payments_terminals_endpoint_url}/activate`

Examples:

`+https://www.server.com/ocpi/ptp/2.2.1/payments/terminals/activate`

Request Body

The request body contains an Terminal to activate.

NOTE	The terminal_id is optional in the activation request as it will be set by the PTP. The cardinality for the remaining fields stays the same.

Type	Card.	Description
> Terminal	1	Terminal object to update.

Example: Activating a Terminal with basic data

```
POST To URL: https://www.server.com/ocpi/ptp/2.2.1/payments/terminals/activate

{
  "terminal_id": "a06dc823-3e5a-40c8-89cf-1b5b9e941412",
  "location_ids": [
    "df37373d-1669-4127-a6ac-d86750095119"
  ],
  "reference": "Term-SerialNumber",
  "last_updated": "2019-12-10T17:16:15Z"
}
```

16.5.1.8. POST Deactivate Terminal Method

This POST request should be used by the CPO to deactivate a given terminal. This may be necessary when the terminal is broken or there is an address change for the given terminal.

Endpoint structure definition:

`{payments_terminals_endpoint_url}/{terminal_id}/deactivate`

Examples:

`+https://www.server.com/ocpi/ptp/2.2.1/payments/terminals/55719888- ed09-4cca-82cc-803bdb77bf26/deactivate`

16.5.1.9. GET Financial Advice Confirmations Method

Fetch Financial Advice Confirmations from a PTP system.

Endpoint structure definition:

`{payments_financial_advice_confirmation_endpoint_url}?[date_from={date_from}]&[date_to={date_to}]&[offset={offset}]&[limit={limit}]`

Examples:

`https://www.server.com/ocpi/ptp/2.2.1/payments/financial-advice-confirmations/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2.1/payments/financial-advice-confirmations/?offset=50`

`https://www.server.com/ocpi/2.2.1/payments/financial-advice-confirmations/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/ptp/2.2.1/payments/financial-advice-confirmations/?offset=50&limit=100`

Request Parameters

If the optional parameters date from and/or date to are provided, only Financial Advice Confirmations with `last_update` between the given `{date_from}` (including) and `{date_to}` (excluding) will be returned.

This request is paginated, it supports the pagination related URL parameters.

Parameter	Datatype	Required	Description
date_from	DateTime	no	Only return Sessions that have `last_updated` after or equal to this Date/Time (inclusive).
date_to	DateTime	no	Only return Sessions that have `last_updated` up to this Date/Time, but not including (exclusive).
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

Response Data

The response contains a list of Financial Advice Confirmation objects that match the given parameters in the request, the header will contain the pagination related headers.

Any older information that is not specified in the response is considered no longer valid. Each object must contain all required fields. Fields that are not specified may be considered as null values.

Datatype	Card.	Description
FinancialAdviceConfirmation	*	List of Financial Advice Confirmation objects that match the request parameters.

16.5.1.10. GET Financial Advice Confirmation Method

If the CPO wants to check the status of a Financial Advice Confirmations object in the PTP system, it might GET the object from the PTP system for validation purposes.

Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
financial_advice_confirmation_id	CiString(36)	yes	Financial Advice confirmation.id of the financial advice confirmation object to retrieve.

Response Data

The response contains the requested object.

Type	Card.	Description
> Financial Advice Confirmation	1	Requested Financial Advice Confirmation.

16.5.2. Receiver Interface

Typically implemented by market roles like: CPO.

16.5.2.1. Terminals Interface

Method	Description
GET	Retrieve a Terminal object from the CPO's system with Terminal.id equal to `{terminal_id}`.
POST	Creating a Terminal object in the CPO's system.

16.5.2.2. Financial Advice Confirmation Interface

Method	Description
GET	Retrieve a Financial Advice Confirmation object from the CPO's system with FinancialAdviceConfirmation.id equal to `{financial_advice_confirmation_id}`.
POST	Creating a Financial Advice Confirmation object in the CPO's system.

16.5.2.3. GET Terminal Method

The PTP system might request the current version of a Terminal object from the CPO's system to, for example, validate the state.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
terminal_id	CiString(36)	yes	id of the Terminal object to get from the CPO's system.

Response Data

The response contains the requested Terminal object.

Datatype	Card.	Description
Terminal	1	Requested Terminal object.

16.5.2.4. POST Terminal Method

The POST should be used by the PTP to create a newly shipped terminal on the CPO's system. Here, if no activation was sent before no location ids should be included as the assignment will be done by the CPO. The object sent here can be just the terminal id or an object with additional data if known through the terminal order and/or activation process.

Request Body

The request contains the new Terminal object.

Type	Card.	Description
Terminal	1	New Terminal object.

Example: Create a minimal Terminal

```
POST To URL: https://www.server.com/ocpi/cpo/2.2.1/payments/terminals/

{
  "terminal_id": "452cf8a1-79aa-4a0e-9aee-dc788586053c"
}
```

Example: Create a Terminal

```
POST To URL: https://www.server.com/ocpi/cpo/2.2.1/payments/terminals/

{
  "terminal_id": "452cf8a1-79aa-4a0e-9aee-dc788586053c",
  "address": "Street 1",
  "city": "Vienna",
  "country": "AUT",
  "coordinates": {
    "latitude": "51.047599",
    "longitude": "3.729944"
  },
  "customer_reference": "ChargePoint",
  "invoice_base_url": "https://somecompany.com/invoices",
  "invoice_creator": "CPO",
  "location_ids": [],
  "last_updated": "2018-12-10T17:16:15Z"
}
```

16.5.2.5. GET Financial Advice Confirmation Method

The PTP system might request the current version of a Financial Advice Confirmation object from the CPO's system to, for example, validate the state.

Request Parameters

The following parameters shall be provided as URL segments.

Parameter	Datatype	Required	Description
financial_advice_confirmation_id	CiString(36)	yes	id of the Financial Advice Confirmation object to get from the CPO's system.

Response Data

The response contains the requested Financial Advice Confirmation object.

Datatype	Card.	Description
FinancialAdviceConfirmation	1	Requested Financial Advice Confirmation object.

16.5.2.6. POST Financial Advice Confirmation Method

The POST should be used by the PTP to create a Financial Advice confirmation on the CPO's system. This will be used to get the status of the capture and also the required eft data to put on the invoice. The PTP has to make sure to use the same authorization reference as provided in the Commands.StartSession so that the CPO can properly map the financial advice to the session.

Request Body

The request contains the new Financial Advice Confirmation object.

Type	Card.	Description
FinancialAdviceConfirmation	1	New Financial Advice Confirmation object.

Example: Create a Financial Advice Confirmation

```
POST To URL: https://www.server.com/ocpi/cpo/2.2.1/payments/financial-advice-confirmations/

{
  "id": "452cf8a1-79aa-4a0e-9aee-dc788586053c",
  "authorization_reference": "pp-100100-1948213567",
  "total_costs": {
    "excl_vat": 4.00,
    "incl_vat": 4.40
  },
  "currency": "EUR",
  "eft_data": [
    "Mastercard",
    "AID: 1234",
    "Crypto: 3456",
    "Nr: **** **** **** 1234",
    "SEQ: 00",
    "Amount: EUR 4.40"
  ],
  "capture_status_code": "SUCCESS",
  "capture_status_message": "Capture successfull at PSP",
  "last_updated": "2018-12-10T17:16:15Z"
}
```

16.6. Object description

16.6.1. *Terminal* Object

The Terminal object describes one physical payment terminal. It is designed primarily to establish a mapping between charge points (locations and/or EVSEs) and payment terminals. The object facilitates the configuration of necessary payment-related data, such as customer reference identifiers and invoice URLs

Property	Type	Card.	Description
terminal_id	CiString(36)	1	Unique ID that identifies a terminal.
customer_reference	CiString(36)	?	This reference will be used to link the terminal to a CSMS. The reference might also be provided via the order process.
party_id	CiString(3)	?	This is an alternative to the customer reference which can be used.
country_code	CiString(2)	?	This is an alternative to the customer reference which can be used.
address	CiString(45)	?	Street/block name and house number if available.
city	CiString(45)	?	City or town.
postal_code	CiString(10)	?	Postal code of the terminal, may only be omitted when the terminal has no postal code.
state	CiString(20)	?	State or province of the location, only to be used when relevant.
country	CiString(3)	?	ISO 3166-1 alpha-3 code for the country of this location.
coordinates	GeoLocation	?	Coordinates of the terminal.
invoice_base_url	URL	?	BaseURL to the downloadable invoice
invoice_creator	InvoiceCreator	?	Describes which party creates the invoice for the eDriver.
reference	CiString(36)	?	Mapping value as issued by the PTP(e.g serial number).
location_ids	CiString(36)	*	List of all locations assigned to that terminal.
evse_uids	CiString(36)	*	List of all EVSEs assigned to that terminal.
last_updated	DateTime	1	Timestamp when this Terminal was last updated (or created).

16.6.1.1. Examples

Simple Terminal example which is newly created

```
{
  "terminal_id": "452cf8a1-79aa-4a0e-9aee-dc788586053c",
  "customer_reference": "Chargepoint",
  "address": "Street 1",
  "city": "Vienna",
  "country": "AUT",
  "coordinates": {
    "latitude": "51.047599",
    "longitude": "3.729944"
  },
  "invoice_base_url": "https://somecompany.com/invoices",
  "invoice_creator": "CPO",
  "location_ids": [],
  "last_updated": "2018-12-10T17:16:15Z"
}
```

Terminal example with assigned locations

```
{
  "terminal_id": "9e94f62c-661b-4afa-b6da-019b58fab9ac",
  "address": "Street 1",
  "city": "Vienna",
  "country": "AUT",
  "coordinates": {
    "latitude": "51.047599",
    "longitude": "3.729944"
  },
  "customer_reference": "BP",
  "invoice_base_url": "https://somecompany.com/invoices",
  "invoice_creator": "PTP",
  "location_ids": [
    "df37373d-1669-4127-a6ac-d86750095119",
    "a06dc823-3e5a-40c8-89cf-1b5b9e941412",
    "55719888-ed09-4cca-82cc-803bdb77bf26"
  ],
  "last_updated": "2018-12-10T17:16:15Z"
}
```

Terminal example with assigned locations and EVSEs

```
{
  "terminal_id": "9e94f62c-661b-4afa-b6da-019b58fab9ac",
  "address": "Street 1",
  "city": "Vienna",
  "country": "AUT",
  "coordinates": {
    "latitude": "51.047599",
    "longitude": "3.729944"
  },
  "customer_reference": "BP",
  "invoice_base_url": "https://somecompany.com/invoices",
  "invoice_creator": "PTP",
  "location_ids": [
    "df37373d-1669-4127-a6ac-d86750095119",
    "a06dc823-3e5a-40c8-89cf-1b5b9e941412",
    "55719888-ed09-4cca-82cc-803bdb77bf26"
  ],
  "evse_uids": [
    "17d5f8ea-8832-454f-aff5-257bc6a25353"
  ],
  "last_updated": "2018-12-10T17:16:15Z"
}
```

16.6.2. *Financial Advice Confirmation* Object

The Financial Advice Confirmation object is utilized to encapsulate the financial details of transactions processed at payment terminals. It correlates payment transactions with charging sessions by using the authorization_reference obtained from the Commands.StartSession, Session, and CDR. This reference ensures that each financial transaction can be accurately mapped to its corresponding charging session. Additionally, the object includes eft_data (Electronic Funds Transfer data), which are mandatory for inclusion on invoices to meet legal and regulatory requirements.

Property	Type	Card.	Description
id	CiString(36)	1	Unique ID that identifies a financial advice confirmation.

Property	Type	Card.	Description
authorization_reference	CiString(36)	1	Reference to the authorization given by the PTP in the Commands.StartSession.
total_costs	Price	1	Real amount that was captured at the PSP. This is a consumer price with VAT.
currency	CiString(3)	1	ISO-4217 code of the currency of this tariff.
eft_data	CiString[1..255]	+	Invoice relevant data from the direct payment.
capture_status_code	CaptureStatusCode	1	Code that identifies the financial advice status.
capture_status_message	CiString[1..255]	?	Message about any error at the financial advice.
last_updated	DateTime	1	Timestamp when this financial advice confirmation was last updated (or created).

16.6.2.1. Examples

Example of a successful capture at the PSP

```
{
  "id": "452cf8a1-79aa-4a0e-9aee-dc788586053c",
  "authorization_reference": "pp-100100-1948213567",
  "total_costs": {
    "excl_vat": 4.00,
    "incl_vat": 4.40
  },
  "currency": "EUR",
  "eft_data": [
    "Mastercard",
    "AID: 1234",
    "Crypto: 3456",
    "Nr: **** **** **** 1234",
    "SEQ: 00",
    "Amount: EUR 4.40"
  ],
  "capture_status_code": "SUCCESS",
  "capture_status_message": "Capture successfull at PSP",
  "last_updated": "2018-12-10T17:16:15Z"
}
```

Example of an unsuccessful capture at the PSP

```
{
  "id": "452cf8a1-79aa-4a0e-9aee-dc788586053c",
  "authorization_reference": "pp-100100-1948213567",
  "total_costs": {
    "excl_vat": 0.00,
    "incl_vat": 0.00
  },
  "currency": "EUR",
  "capture_status_code": "FAILED",
  "capture_status_message": "Capture unsuccessful at PSP",
  "last_updated": "2018-12-10T17:16:15Z"
}
```

16.7. Data types

16.7.1. InvoiceCreator *enum*

Value	Description
CPO	The CPO issues the invoice and provides it via the invoice_base_url + authorization_reference.
PTP	The PTP issues the invoice and directly shows/provides it the eDriver via the payment terminal.

16.7.2. CaptureStatusCode *enum*

This enumeration describes the status of the payment capture process following a transaction at an EV charging station. It helps determine the outcome of the transaction and facilitates accurate financial reporting and customer billing.

Value	Description
SUCCESS	Indicates that the payment capture was completed successfully without any issues. Funds were secured and will be settled according to the payment processor's timeline. This status confirms that all checks (e.g., fraud, card validation) passed and the transaction was approved..
PARTIAL_SUCCESS	Used when only part of the transaction amount was approved or when certain conditions of the payment were altered during processing. This might occur in scenarios where the available balance was insufficient for the full requested amount, or specific transaction limits were enforced by the card issuer.
FAILED	Indicates that the payment capture attempt was unsuccessful. This failure can be due to various reasons such as insufficient funds, card expiration, network issues, or refusal by the card issuer.

17. Types

17.1. class

When a data type is defined as a "class" in the OCPI specification, we mean a type whose possible values are sets of zero or more pairs of a string and another value. The string is known as a "key", "field name", or "property", and the value associated with the key is known as a field value. For each class type, the specification lists which strings are required and allowed to occur as field names in values of that type, and what the types of the field values of these fields should be.

In the serialized JSON form of OCPI messages, class values are serialized as JSON objects.

17.2. enum

When a data type is defined as an "enum" in the OCPI specification, we mean a type whose possible values are a finite number of strings.

This type is used for class fields where it is clear that there is only a finite set of possible values that is completely known at the time of writing of the specification. An example of a place where this is used is a class field whose possible values are the days of the week.

In the serialized JSON form of OCPI messages, enum values are serialized as JSON strings.

17.3. OpenEnum *type*

The OpenEnum type is meant for class fields for which the set of all possible values is not known at the time of writing of the specification, but where there are a finite number of known possible values. In this case we want to specify how OCPI implementers can use the known possible values, but also leave room for them to use other values.

This is used for example for connector types, where all implementers should use the same value to identify a widely used connector type like the Type 2 "Mennekes" plug, but where there should also be room for implementers to name new or custom plug types that were not taken into account by OCPI's authors.

In the serialized JSON form of OCPI messages, OpenEnum values are serialized as JSON strings.

When naming new OpenEnum values, OCPI implementers SHOULD follow the "Recommendations for Creators of New Parameters" found in IETF RFC 6648, and SHOULD consult EV Roaming Foundation's guidance on extending OCPI at https://evroaming.org/extending-ocpi/.

17.4. CiString *type*

Case Insensitive String. Only printable ASCII allowed. (Non-printable characters like: Carriage returns, Tabs, Line breaks, etc are not allowed)

17.5. DateTime *type*

All timestamps are formatted as string(25) following RFC 3339, with some additional limitations.

All timestamps SHALL be in UTC. The absence of the timezone designator implies a UTC timestamp. Fractional seconds MAY be used.

Example of how timestamps shall be formatted in OCPI, other formats/patterns are not allowed:

```
2015-06-29T20:39:09Z
2015-06-29T20:39:09
2016-12-29T17:45:09.2Z
2016-12-29T17:45:09.2
2018-01-01T01:08:01.123Z
2018-01-01T01:08:01.123
```

NOTE +00:00 is not the same as UTC.

17.6. DisplayText *class*

Property	Type	Card.	Description
language	string(2)	1	Language Code ISO 639-1.
text	string(512)	1	Text to be displayed to a end user. No markup, html etc. allowed.

Example:

```
{
  "language": "en",
  "text": "Standard Tariff"
}
```

17.7. number *type*

Numbers in OCPI are formatted as JSON numbers. Unless mentioned otherwise, numbers use 4 decimals and a *sufficiently large amount* of digits.

17.8. Price *class*

Property	Type	Card.	Description
before_taxes	number	1	Price/Cost excluding taxes.
taxes	TaxAmount	*	All taxes that are applicable to this price and relevant to the receiver of the Session or CDR.

17.9. TaxAmount *class*

Property	Type	Card.	Description
name	string	1	A description of the tax. In countries where a tax name is required like Canada this can be something like "QST". In countries where this is not required, this can be something more generic like "VAT" or "General Sales Tax".
account_number	string	?	Tax Account Number of the business entity remitting these taxes. Optional as this is not required in all countries.
percentage	number	?	Tax percentage. Optional as this is not required in all countries.
amount	number	1	The amount of money of this tax that is due.

17.10. Role *enum*

Value	Description
CPO	Charge Point Operator Role.
EMSP	eMobility Service Provider Role.
NAP	National Access Point Role (national Database with all Location information of a country).
NSP	Navigation Service Provider Role, role like an eMSP (probably only interested in Location information).
OTHER	Other role.
SCSP	Smart Charging Service Provider Role.

17.11. string *type*

Case Sensitive String. Only printable UTF-8 allowed. (Non-printable characters like: Carriage returns, Tabs, Line breaks, etc are not allowed)

All strings in messages and enumerations are case sensitive, unless explicitly stated otherwise.

17.12. URL *type*

An URL a string(255) type following the w3.org spec.

18. Changelog

18.1. Changes between 2.2.1-d2 and 2.3.0

- Make OCPI Extensible: possible to add modules, fields, enum values for certain enums
- Add a Parking object linked to EVSEs, indicating vehicle type among other properties
- Add a field to the EVSE object to indicate which eMSPs' contracts are accepted
- Add a field to the Location object for a support telephone number
- Information for people with disabilities
- Support for North American taxes
- Take straightforward enum values from the OCPI 3.0 draft, including those that signal 15118 compatibility
- Add new field in Credentials to give hub party ID and make hub clients be reported as normal credentials roles
- Add new Payments module

18.2. Changes between 2.2.1 and 2.2.1-d2

- Removed note that advised against sharing Locations on which home charging reimbursement happens
- Lots of editing and rewriting of Tariffs and step_size documentation
- Updated examples and diagrams to use convention of not using trailing slashes on URLs
- Updated example of a short finished session so that the total energy matches the energies of the charging periods
- Removed stipulation that all charging_periods have a different Tariff from CDRs module, which contradicts other statements in the CDRs module description
- Add a note clarifying Base64 usage in the Authorization header and use more precise wording to specify the Base64 encoding
- Replaced 2.2 by 2.2.1 in example URLs and a few other places where 2.2 was used to mean the current version.
- Added missing forward slash in Tokens Receiver interface endpoint URL structure definition
- Changed "Tariff Elements" to "Tariffs" in description of tariffs field in CDR object definition
- Added country_code and party_id to CdrToken examples
- Changed country_id to country_code in credentials explanation
- Replaced copy-pasted text about charging profiles in ChargingPreferencesResponse description
- Replaced "cpo" by "emsp" in example URL of receiver-side session module
- Allowed eMSP to replace CPO-issued session IDs when exchanging charging profiles with an SCSP
- Changed text to give more actionable advice on how to set the Interface role in the Endpoint object for the credentials module
- Added an explanation of why the Tariffs module doesn’t say anything about price rounding
- Added a note that CPOs should avoid using physical hardware ids for EVSE.uid
- Removed example about a free hour of parking that conflicts with spec

- Changed "GET" to "PUT" in sequence diagram showing routing header usage with Broadcast Push

18.3. Changes between OCPI 2.2 and 2.2.1

Lots of typos fixed and textual improvements.

The following changes to messages/objects etc.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Status codes	Minor / Minor	Minimal / Minimal	Added 4000 Hub generic error
CDRs / CdrToken Class	Minor / Minor	Minor / Major	Added mandatory `country_code` and `party_id` fields to make sure that a CdrToken uniquely identifies a charge token
CDRs / CDR Object	Minor / Minor	Minimal / Minimal	Added `home_charging_compensation` optional field.
CDRs / CdrLocation Class	Minor / Minor	Minimal / Minimal	- Changed `postal_code` to optional, inline with the Location module. - Added `state` as optional field, inline with the Location module.
CDRs / SignedData Class	Major / Major	Minimal / Minimal	Changed `public_key` to string, was CiString
CDRs / SignedValue Class	Major / Major	Minimal / Minimal	- Increased `signed_data` length from 512 to 5000. - Changed `plain_data` to string, was CiString - Changed `signed_data` to string, was CiString - Changed `url` to string, was CiString
Commands / StartSession Object	Minor / Minor	Medium / Minimal	Added optional field: `connector_id` to support OCPP 1.x Charge Points with multiple connectors per EVSE.
Locations / Capability Enum	Minor / Minor	Medium / Minimal	Added `START_SESSION_CONNECTOR_REQUIRED` to support OCPP 1.x Charge Points with multiple connectors per EVSE.
Locations / ConnectorType Enum	Minor / Minor	Minimal / Minimal	Added NEMA, GB/T, ChaoJi and Domestic M, N and O connector types.
Locations / PowerType Enum	Minor / Minor	Minimal / Minimal	Added `AC_2_PHASE` an `AC_2_PHASE_SPLIT` to the enum to support two phase chargers.

18.4. Changes between OCPI 2.1.1 and 2.2

Lots of typos fixed and textual improvements.
Improved/fixed all descriptions and examples with relation to the Tariff `step_size`.

The following changes to messages/objects etc.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
CDRs / CDR Object	Medium / Major	Average / Large	- Added `county_code` and `party_id` fields, making it easier to determine the owner of a CDR. - Added `session_id` field, making it easier to match a CDR to a Session. - Renamed `stop_date_time` field to `end_date_time`, to bring the naming inline with the rest of OCPI. - Changed `total_cost` field from type: `number` to `Price`, this provides the eMSP with the total cost including VAT. - Replaced `auth_id` field with `CdrToken`. `auth_id` alone could not be used to uniquely identify a Token. By copying the information for the dynamic Token object, the CDR will always reflect the 'true' status of Token at the start of the charging session. - Replaced `location` field with `cdr_location`, this also changed type, from `Location` to `CdrLocation`. Reusing the `Location` object always caused a lot of confusing, things were not clear. By creating a dedicated object `CdrLocation` with only the relevant fields, things should be much clearer. - Added `credit` and `credit_reference_id` fields, to allow for Credit CDRs to be send. - Added `total_fixed_cost`, `total_energy_cost`, `total_time_cost`, `total_parking_cost` and `total_reservation_cost` fields, to allow more cost details in the CDRs. - Added `authorization_reference` field for binding an authorization to the resulting session. - Added `signed_data` field, enabling OCPI to be used to transport signed meter data from the Charge Point to the eMSP and EV driver, can be used for Eichrecht. - Added `invoice_reference_id` field (optional), to allow a CDRs to reference an invoice. - Field `id` changed in length from 36 to 39, to allow for something to be appended after the original id in case of a Credit CDR.
Commands / AuthMethod enum	Minor / Minor	Minimal / Minimal	Added `COMMAND` value, to enable reporting authorization via Command like: StartSession or ReserveNow.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
CDRs / CdrDimensionType enum	Medium / Medium	Minimal / Minimal	Added `RESERVATION_TIME` value, to enable reporting of cost for a reservation in a Session or CDR.
CDRs / CdrDimensionType enum	Medium / Medium	Minimal / Minimal	Removed `FLAT` value, that is of no use on a CDR, only causes confusion, should have been removed when CdrDimensionType was created from DimensionType of OCPI 2.0.
CDRs / ChargingPeriod class	Medium / Medium	Minimal / Minimal	Added `tariff_id` field to ChargingPeriod, when the session switches from one tariff to another, this needs to be known, can be relevant with Preference based Smart Charging.
ChargingProfiles	Major / Major	Large / Large	Added new ChargingProfiles module.
Commands / CancelReservation Object	Minor / Minor	Minimal / Minimal	Added CancelReservation object for the cancel reservation command.
Commands / CommandType Enum	Minor / Minor	Minimal / Minimal	Added `CANCEL_RESERVATION` value, adding the cancel reservation command.
Commands / CommandResponse Object	Minor / Minor	Minimal / Minimal	- Added `message` field, enables the CPO to send a message to the user when something goes wrong. - Added `timeout` field, enables the eMSP to cleanup not responded outstanding commands.
Commands / ReserveNow Object	Minor / Medium	Minimal / Average	- Changed `location_id` and `evse_uids` from string to CiString, making them case-insensitive, which had always been the idea. Lengths changed from 39 to 36, matching changes in the object definitions. - Changed `reservation_id` from int to CiString(36), making it possible to use UUIDs. - Added `authorization_reference` field for binding an authorization to the resulting session. - Changed/added requirements in description of ReserveNow Object.
Commands / StartSession Object	Minor / Medium	Minimal / Average	- Changed `location_id` and `evse_uids` from string to CiString, making them case-insensitive, which had always been the idea. Lengths changed from 39 to 36, matching changes in the object definitions. - Added `authorization_reference` field for binding an authorization to the resulting session. - Changed/added requirements in description of StartSession Object.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Commands / StopSession Object	Minor / Minor	Minimal / Minimal	Changed `session_id` from string to CiString, making it case-insensitive, which had always been the idea.
Commands / UnlockConnector Object	Minor / Minor	Minimal / Minimal	- Changed `location_id`, `evse_uids` and `connector_ids` from string to CiString, making them case-insensitive, which had always been the idea. - Length of `location_id` and `evse_uids` changed from 39 to 36, matching changes in the object definitions.
Commands / CommandResponseType Enum	Minor / Minor	Minimal / Minimal	removed `TIMEOUT` as possible value. This is moved to the new CommandResult object.
Commands / CommandResultType Enum	Minor / Minor	Minimal / Minimal	added `CANCELED_RESERVATION` value. Make it possible for a CPO to cancel an existing reservation in case of issues with the Charge Point.
Commands / CommandResult Object	Medium / Medium	Medium / Medium	Changed result message from CPO to eMSP from CommandResponse to CommandResult to make it more clear.
Credentials / Credentials Object	Minor / Minor	Minimal / Minimal	Changed `country_code` and `party_id` from string to CiString, making them case-insensitive, which had always been the idea. Replaced the `business_details`, `party_id` and `country_code` field with a `roles` list. Making it possible to implement different parties and roles in the same OCPI instance. The fields are now moved into a new `CredentialsRole class.
HubClientInfo	Medium / Medium	Medium / Medium	Added new HubClientInfo module.
Locations / Sender GET Object method	Minor / Minor	Minimal / Minimal	- Changed `location_id`, `evse_uids` and `connector_ids` from string to CiString, making them case-insensitive, which had always been the idea. - Length of `location_id` and `evse_uids` changed from 39 to 36, matching changes in the object definitions.
Locations / Receiver GET & PUT methods	Minor / Minor	Minimal / Minimal	- Changed `country_code`, `party_id`, `location_id`, `evse_uids` and `connector_ids` from string to CiString, making them case-insensitive, which had always been the idea. - Length of `location_id` and `evse_uids` changed from 39 to 36, matching changes in the object definitions.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Locations / Receiver PATCH method	Minor / Minor	Minimal / Minimal	- Changed `country_code`, `party_id`, `location_id`, `evse_uids` and `connector_ids` from string to CiString, making them case-insensitive, which had always been the idea. - Length of `location_id` and `evse_uids` changed from 39 to 36, matching changes in the object definitions. - Added description on how to handle `last_updated`, and made it required for all PATCH requests.
Locations / Connector Object	Minor / Minor	Minimal / Minimal	- Field `id` is changed from string to CiString, making it now case-insensitive, which had always been the idea. - Added `max_electric_power` field, some DC Fast Charger have a lower max power then can be calculated form `voltage` and `amperage`. - Changed `tariff_id` field to `tariff_ids`, and changed cardinality from `?` to `*`. Making it possible to make provided tariffs for different Smart Charging Preferences and also for ad hoc payment. Changed type from string to CiString, matching the change to Tariff.id. - Changed `amperage` field to `max_amperage` and `voltage` field to `max_voltage`, to better reflect the real meaning of both fields.
Locations / EVSE Object	Minor / Minor	Minimal / Minimal	- Fields `uid` and `evse_id` is changed from string to CiString, making them case-insensitive, which had always been the idea. - length of `uid` changed from 39 to 36, as 36 is enough to store UUID and GUIDs.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Locations / Location Object	Minor / Minor	Minimal / Minimal	- Added `county_code` and `party_id` fields, making it easier to determine the owner of a Location. - Field `id` is changed from string to CiString, making it now case-insensitive, which had always been the idea. - length changed from 39 to 36, as 36 is enough to store UUID and GUIDs. - Added `state` field, optional, to allow as much different address schemes from around the world as possible.. - Changed `postal_code` field from required to optional, with the remark that omitting is only allowed when location has no postal_code. - Changed `time_zone` field from optional to required, as the opening hours and tariff start/end depend on this, they are more and more important. - Renamed `type` field to `parking_type` and made it optional. It better reflects what this field really describes. - Added `publish` field, required, to control which locations may or may not be publish in apps etc. - Added `publish_allowed_to` field, optional, to give access to locations to only a limited set of users.
Locations / AdditionalGeoLocation class	Minor / Minor	Minimal / Minimal	Changed regex for fields: `latitude` and `longitude` from fixed 6 decimal places, to more flexible 5 to 7 decimal places.
Locations / Capability enum	Minor / Minor	Minimal / Minimal	added new values for: `CHARGING_PREFERENCES_CAPABLE`, `DEBIT_CARD_PAYABLE` and `TOKEN_GROUP_CAPABLE`.
Locations / ConnectorType enum	Minor / Minor	Minimal / Minimal	added new values for: `PANTOGRAPH_TOP_DOWN` and `PANTOGRAPH_BOTTOM_UP`.
Locations / EnvironmentalImpact class	Minor / Minor	Minimal / Minimal	Changed field name from `source` to `category`, this was a copy/past error in an older version of OCPI, as this is not used (much) yet, it is better for understandability of OCPI for correct the field name.
Locations / Facility enum	Minor / Minor	Minimal / Minimal	added new values for: `BIKE_SHARING`, `PARKING_LOT`, `TRAM_STOP` and `METRO_STATION`.
Locations / GeoLocation class	Minor / Minor	Minimal / Minimal	Changed regex for fields: `latitude` and `longitude` from fixed 6 decimal places, to more flexible 5 to 7 decimal places.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Locations / Hours class	Minor / Minor	Minimal / Minimal	removed to option for either: twentyfourseven or regular_hours, now twentyfourseven is always required and regular_hours is required when twentyfourseven=false, this is much less confusing.
Locations / Image class	Minor / Minor	Minimal / Minimal	Changed field `type` from string to CiString, is for machine to machine communication, so UTF-8 is not needed.
Locations / RegularHours class	Minor / Minor	Minimal / Minimal	Improved the regex for time format.
Locations / LocationType enum	Minor / Minor	Minimal / Minimal	Renamed to: `ParkingType` Added the values: `ON_DRIVEWAY` and `ALONG_MOTORWAY` Removed the values: `OTHER` and `UNKNOWN`, no longer needed as this is now optional.
Sessions / Sender PUT method	Medium / Medium	Large / Large	Added setting Charging Preferences on a session. Proving the CPO with preferences from the driver, needed for Smart Charging. For this the following data types are added: ChargingPreferences, ChargingPreferencesResponse, ProfileType,
Sessions / Receiver GET and PUT methods	Minor / Minor	Minimal / Minimal	Changed `country_code`, `party_id` and `session_id` from string to CiString, making them case-insensitive, which had always been the idea.
Sessions / Receiver PATCH method	Minor / Minor	Minimal / Minimal	Changed `country_code`, `party_id` and `session_id` from string to CiString, making them case-insensitive, which had always been the idea. Added description and requirements how to add `charging_periods` and made `last_updated` required for all PATCH requests.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Sessions / Session Object	Minor / Medium	Minimal / Average	- Added `county_code` and `party_id` fields, making it easier to determine the owner of a Session. - Field `id` is changed from string to CiString, making it now case-insensitive, which had always been the idea. - Changed `total_cost` field from type: `number` to `Price`, this provides the eMSP with the total cost including VAT. - Added `start_date_time` and `end_date_time` fields. - Replaced `auth_id` with `CdrToken` class. `auth_id` alone could not be used to uniquely identify a Token. - Replaced `location` object with `location_id`, `evse_uid` and `connector_id`. Having the `Location` Object in the `Session` was overkill, only reference is more inline with the rest. - Added `authorization_reference` field for binding an authorization to the resulting session.
Tariffs / Receiver PATCH method	Minor / Minor	Minimal / Minimal	PATCH is removed from Tariffs as this was seen is not useful, use PUT instead.
Tariffs / Tariff Object	Minor / Minor	Minimal / Minimal	- Added `county_code` and `party_id` fields, making it easier to determine the owner of a Tariff. - Field `id` is changed from string to CiString, making it now case-insensitive, which had always been the idea. - Renamed `start_datetime` field to `start_date_time`, to bring the naming inline with the rest of OCPI. - Renamed `end_datetime` field to `end_date_time`, to bring the naming inline with the rest of OCPI. - Added optional `min_price` field, making it possible to set a minimum price on a Charging Session. - Added optional `max_price` field, making it possible to set a maximum price on a Charging Session. - Added `type` field to make it possible to make different tariffs for different Smart Charging Preferences and also for ad hoc payment.
Tariffs / PriceComponent class	Minor / Minor	Minimal / Minimal	- Added `vat` field to send the applicable VAT with every tariff component.
Tariffs / ReservationRestrictionType enum	Minor / Minor	Minimal / Minimal	Added new enum for Reservation restrictions.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Tariffs / TariffRestrictions class	Minor / Minor	Minimal / Minimal	Added optional `reservation` field, making it possible to define the tariff of a reservation (and an expired reservation). Added optional `min_current` and `max_current` field, making it possible to have a tariff that depends on the current being charged, instead of the power. Improved the regex for time format.
Tokens / Sender GET & POST methods	Minor / Minor	Minimal / Minimal	Changed `country_code`, `party_id` and `token_uid` from string to CiString, making them case-insensitive, which had always been the idea.
Tokens / Receiver GET & PUT methods	Minor / Minor	Minimal / Minimal	Changed `country_code`, `party_id` and `tariff_id` from string to CiString, making them case-insensitive, which had always been the idea. Added `token_type` field, making it possible to make a distinction between different Token types with the same `uid`.
Tokens / Receiver PATCH method	Minor / Minor	Minimal / Minimal	Changed `country_code`, `party_id` and `tariff_id` from string to CiString, making them case-insensitive, which had always been the idea. Added `token_type` field, making it possible to make a distinction between different Token types with the same `uid`. Made `last_updated` required for all PATCH requests.
Tokens / Token Object	Minor / Minor	Minimal / Minimal	- Added `county_code` and `party_id` fields, making it easier to determine the owner of a Token. - Fields `uid` changed from string to CiString, making it now case-insensitive, which had always been the idea. - Fields `auth_id_` renamed to `contract_id`, a much more logical and less confusing name. Also changed from string to CiString, making it now case-insensitive, which had always been the idea. - Added `group_id` field to enable support for OCPP GroupId/ParentId. - Added `default_profile_type` field to enable a default Preference base Smart Charging ProfileType to be provided for a user. - Added `energy_contract` field to make it possible, if allowed, to use a drivers energy supplier/contract at a Charge Point.
Tokens / AuthorizationInfo Object	Minor / Medium	Minimal / Average	Added `token` field to enable real-time authorization of unknown Tokens. Added `authorization_reference` field for binding an authorization to the resulting session.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Tokens / LocationReferences class	Minor / Minor	Minimal / Minimal	- Changed `location_id` and `evse_uids` from string to CiString, making them case-insensitive, which had always been the idea. - Length of `location_id` and `evse_uids` changed from 39 to 36, matching changes in the object definitions. - Removed `connector_ids`, this was not usable as they are not unique within the Location, there is also no use case.
Tokens / TokenType enum	Minor / Minor	Minimal / Minimal	Added value `AD_HOC_USER` and APP_USER`. As more and more eMSPs are launching Apps, this becomes more common, so a special categories are useful.
Versions / Endpoint class	Medium / Medium	Minimal / Minimal	Field `role` added, making it possible to have one OCPI version end-point for both eMSP and CPO role, so one OCPI connection when both CPO and eMSP implemented by the same party.
Transport & Format	Medium / Medium	Medium / Medium	To enable routing of messages through a Hub, new 'OCPI-to-' and 'OCPI-from-' headers are introduced.
Transport & Format	Minor / Minor	Minimal / Minimal	Unique message ID and Correlation message ID headers are now required in every request/response.
Types / DateTime	Minor / Minor	Minimum / Minimum	Changed to: RFC 3339 (was ISO 8601) this does not change the OCPI format, RFC 3339 is more limited, and therefor more inline with OCPI then ISO 8601 was. Fractional seconds are now allowed.
Types / string	Minor / Minor	Minimum / Minimum	Type string changed from ASCII to UTF-8. String is used for human-readable information and thus needed to support for a lot more character sets then only ASCII.