OCPI 2.2

Open Charge Point Interface

document version 2.2-RC1, 04-02-2019
github.com/ocpi

Table of Contents

Version History

Version	Date	Author	Description
2.2 RC1	04-02-2019	Robert de Leeuw *IHomer*	Release Candidate 1 of OCPI 2.2: Adds Smart Charging Profiles, Hub Support, Credit CDRs and some more minor changes. See changelog
2.2 DRAFT1	10-12-2018	Robert de Leeuw *IHomer*	First draft of OCPI 2.2: contains lots of minor changes, some new features, but still misses Smart Charging Profiles and Hub Support. Proposals for the last 2 are ready, but still need some work and need to be integrated into this document. See changelog
2.1.1	08-06-2017	Robert de Leeuw *IHomer*	Fixed 4 bugs found in OCPI 2.1, lots of small textual improvements: see changelog
2.1	08-04-2016	Robert de Leeuw *IHomer*	Added command module. Added support for real-time authorization. Lots of small improvements: see changelog
2.0-d2	15-02-2016	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	30-12-2015	Robert de Leeuw *IHomer* Simon Philips *Becharged* Chris Zwirello *The New Motion* Simon Schilling	First official release of OCPI.
0.4	04-11-2014	Olger Warnier *The New Motion*	First draft of OCPI. (Also known as Draft v4)
0.3	06-05-2014	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 then OCPI 2.1.
- OCPI 2.0-d2 is the same protocol version as OCPI 2.0, but a newer documentation revision: same protocol, newer documentation.

1. OCPI

1.1. OCPI 2.2

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

1.1.1. Editorial note

This is OCPI 2.2 Release Candidate 1. Functional everything is in this release candidate. The first draft has not had a lot of review, so we are not yet confident enough to release a FINAL version yet. We hope that people take the time to review this version and sand feedback so we can improve this. Even if you do not find any issues, please let us know.

We encourage companies to start implementing OCPI 2.2 based on this release candidate. The proof is in the pudding, we need implementations to proof OCPI 2.2 is mature enough to be released as FINAL. When developers are implementing a specification they are the once finding real world issues, most of the times. If they are able to implement OCPI 2.2 and get it working with other parties, that is the proof we need.

We still want to add more examples, some sequence diagrams and use cases to explain the new functionality. We hope to add these in the coming weeks.

1.1.2. Changes/New functionality:

- Support for Hubs
    - Message routing headers
    - Hub Client Info
- Support for Virtual CPO's and eMSP's and other roles
- Charging Profiles
- Preference based Smart Charging
- Improvements:
    - CDRs: Credit CDRs, VAT, Session_id, CdrLocation, CdrToken
    - Sessions: VAT, CdrToken
    - CDRs: Tariff types, Min/Max price, reservation tariff, Much more examples
    - Locations: Multiple Tariffs, Lost of small improvements
    - Tokens: Group_id, energy contract
    - Commands: Cancel Reservation added

For more information on detailedchanges see changelog.

1.2. 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, finally, the exchange of smart-charging commands 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 functionalities): * A good roaming system (for bilateral usage and/or via a 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 station without pre-registration.

Starting in 2009, e-laad foundation and the predecessor of the eViolin association specified 2 standards in order to retrieve charge point details and active state. These are called the VAS interface and the Amsterdam interface. In this same period, a CDR format for the exchange of charge sessions between eViolin members was defined. This format is currently in use by the majority of the eViolin members. (eViolin is the branch organisation for EV operators and service providers in NL and responsible for national roaming and issuing of ID's). This resulted in 2014 in the development of OCPI.

An international group of companies already supports OCPI. Initiators are EV Box, The New Motion, ElaadNL, BeCharged, Greenflux and Last Mile Solutions. Other participants include Next Charge, Freshmile, Plugsurfing, Charge-partner, Hubject, e-clearing.net, IHomer and Siemens. Several other major organizations and roaming platforms are interested in participating. The Netherlands Knowledge Platform for Charging Infrastructure (NKL) facilitates and coordinates this project to guarantee progress and ensure development and results. Part of this project is to find a place to continue development in the future.

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 is taken into account and some elements have been updated to match nowadays use.

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

2. Terminology and Definitions

2.1. Abbreviations

Abbr.	Description
OCPI	Open Charge Point Interface
OCPP	Open Charge Point Protocol
CDR	Charge Detail Record
CPO	Charging Point Operator
eMSP	e-Mobility Service Provider

2.2. Provider and Operator abbreviation

In OCPI it is advised to use eMI3 compliant names for Contract IDs and EVSE IDs. The provider and the operator name is important here, in order to target the right provider or operator, they need to be known up front, 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. At this moment, the following countries have an authority that keeps track of the known providers and operators:

2.2.1. The Netherlands

The Dutch foundation, named eViolin keeps the registry for The Netherlands.

- The list of operator IDs and provider IDs can be viewed on their website eViolin/Leden.

2.2.2. Germany

The BDEW organisation keeps the registry for Germany in their general code number service bdew-codes.de.

- Provider ID List See https://bdew-codes.de/Codenumbers/EMobilityId/ProviderIdList
- EVSE Operator ID List See https://bdew-codes.de/Codenumbers/EMobilityId/OperatorIdList

2.2.3. Austria

Austrian Mobile Power GmbH maintains a registry for Austria. This list is not publicly available. For more information visit austrian-mobile-power.at

2.2.4. France

The AFIREV* organisation will keep/keeps the registry for France. It provides operation Id for CPO and eMSP in compliance with eMI3 id structure. The prefix of these Ids is the “fr” country code. AFIREV will also be in charge of the definition of EVSE-Id structure, Charging-Pool-Id structure (location), and Contract-Id structure for France. AFIREV bases its requirements and recommendations on eMI3 definitions.

AFIREV stands for: Association Française pour l’Itinérance de la Recharge Électrique des Véhicules

2.3. 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.

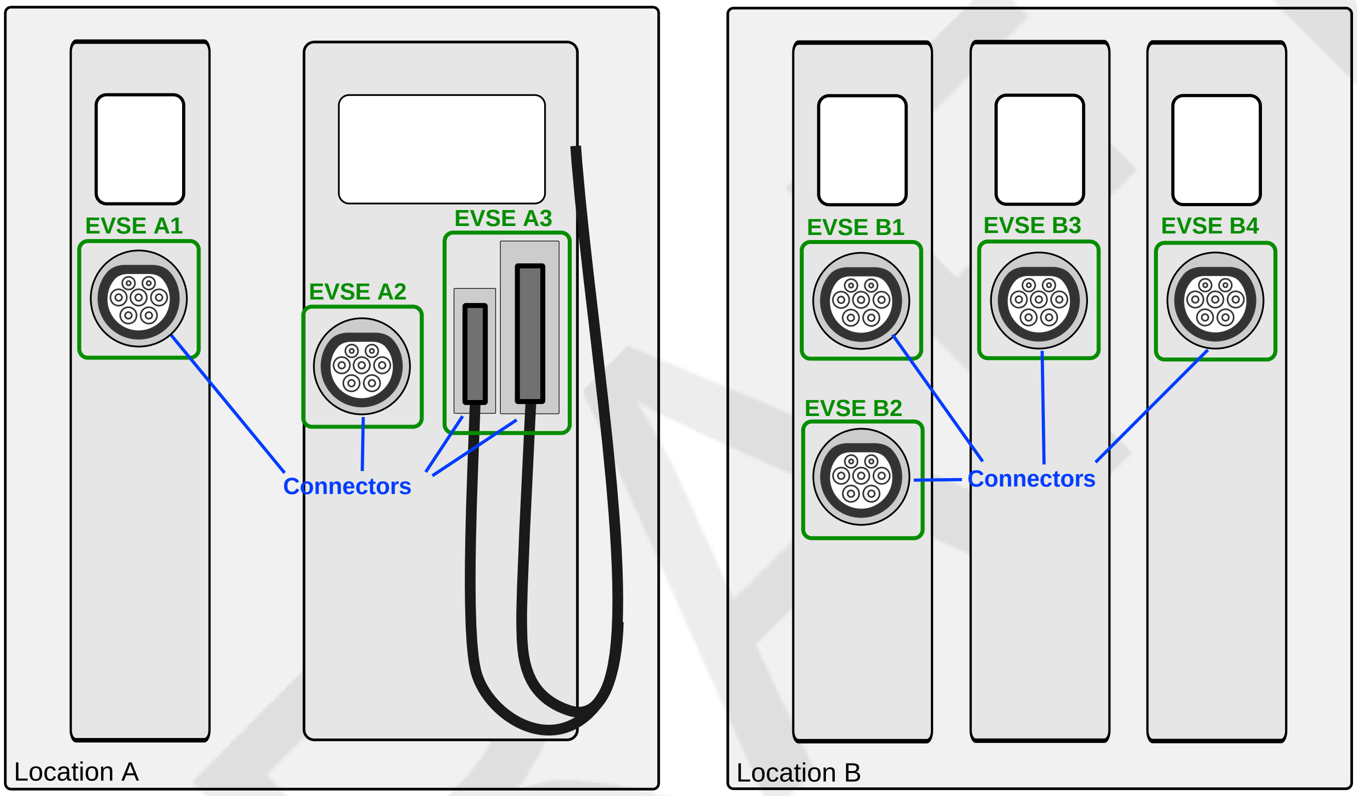

Figure 1. Topology

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).

2.4. Variable names

In order 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 space.

2.5. 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 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]

3. Transport and format

3.1. JSON / HTTP implementation guide

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

3.1.1. Security and authentication

The interfaces are protected on 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 in order to setup a secure SSL connection.

3.1.2. Authorization header

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

```
Authorization: Token IpbJOXxkxOAuKR92z0nEcmVF3Qw09VG7I7d/WCg0koM=
```

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' then 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'.

Its parameter is a string consisting of printable, non-whitespace ASCII characters.

The credentials token must uniquely identify the requesting party. This way, the server can use the information in the Authorization header to the 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 a HTTP `401 - Unauthorized` status code.

When a server receives a request with a valid `CREDENTIALS_TOKEN_A`, on another module then: `credentials` or `versions`, the server SHALL respond with a HTTP `401 - Unauthorized` status code.

3.1.3. Pull and Push

OCPI supports both 'pull' and 'push' models.

- Push: Changes in objects, and new objects are send (semi) real-time to receiver.
- Pull: Receiver request a (full) list of objects every X times.

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 a number of clients are requesting long lists frequently. 'Push' implementation tend to use much less resources. It is therefor 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).

3.1.4. Request format

The request method can be any of GET, 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.

3.1.4.1. 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 amount of objects returned in the response to a GET request, while still enabling the client to retrieve all objects by doing multiple request with different parameters. Without pagination the server had 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	Description
offset	The offset of the first object returned. Default is 0 (the first object).
limit	Maximum number of objects to GET. Note: 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.

Example: With offset=0 and limit=10 the server shall return the first 10 records (if 10 objects match the request). Then 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 it is important that the sequence of objects does not change. (or as little as possible) It is best practice to return the oldest (by creation date, not the `last_updated` field) first. 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. But the client doesn't have to correct for this. Only the last page will be different (or an additional page). So 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 update. The client detects this: `X-Total-Count` will be lower in a next request. It is advised redo the previous GET but then with the `offset` lowered by 1 (if the `offset` was not 0) and after that continue crawling the 'next' page links. When an object before this page has been updated, then the client has missed 1 object.

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

HTTP Header	Description
Link	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 example below.
X-Total-Count	(Custom HTTP Header) Total number of objects available in the server system that match the give 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 then the total number of CDRs available in the CPO system.
X-Limit	(Custom HTTP Header) Number of objects that are returned. 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.

Pagination Examples

Example of a required OCPI pagination link header:

```
Link: <https://www.server.com/ocpi/cpo/2.2/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/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/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/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/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/cdrs/?offset=100&limit=100>; rel="next"
```

3.1.4.2. 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.

3.1.4.3. 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 mimetype 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.

3.1.5. 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 send 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 and party_id of the other party are received during the credentials handshake, so that a server might know the values a client will use in an URL.

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/tariffs/NL/TNM/14
```

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

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

To identified 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.

3.1.5.1. Errors

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. A Server implementation is advised to return an OCPI status code: 2001.

3.1.6. 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	Response code, as listed in Status Codes, indicates how the request was handled. To avoid confusion with HTTP codes, at least four digits are used.

Property	Type	Card.	Description
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, an HTTP error response code is expected, including the response format above, that contains more details. HTTP status codes are described on w3.org.

3.1.6.1. 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"
}
```

3.1.6.2. Example: Version details response (one object)

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

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

```
{
  "data": {
    "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"
}
```

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

```
{
  "data": [{
    "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"
  }, {
    "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"
  }, {
    "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"
}
```

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

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

3.1.7. Message Routing Headers

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

With the introduction of Message Routing, OCPI is now better usable for communication via Hubs. With this same functionality is also becomes possible to implement virtual CPO/eMSPs. A Virtual party is a party that does not has its own back-office, but relies on another parties IT systems. This is sometimes also called white-label CPO/eMSP.

When OCPI is used to communicatie via a Hub or to a virtual CPO/eMSP the following 4 HTTP headers can be used, in requests and responses.

HTTP Header	Description
OCPI-to-party-id	'party id' of the connected party this messages is to be send to.
OCPI-to-country-code	'country code' of the connected party this messages is to be send to.

HTTP Header	Description
OCPI-from-party-id	'party id' of the connected party this messages is send from.
OCPI-from-country-code	'country code' of the connected party this messages is send from.

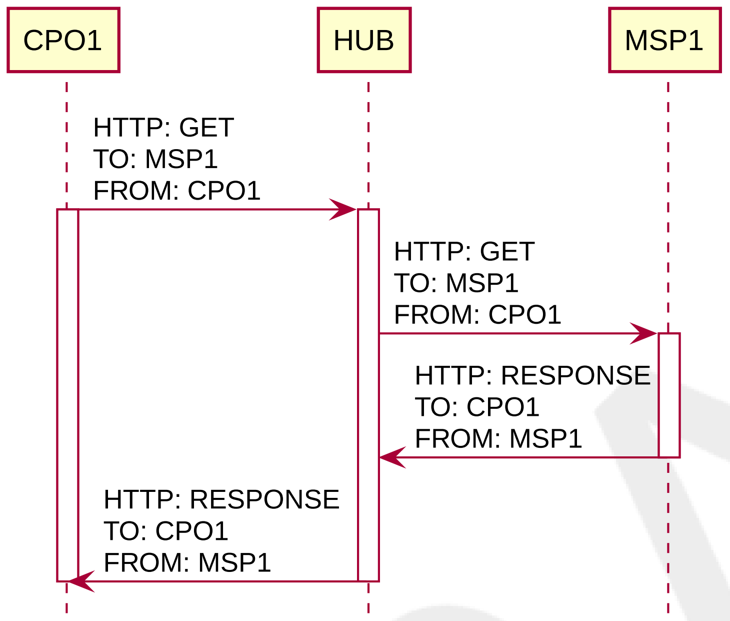

Figure 2. Example sequence diagram of a GET for 1 Object from a CPO to an MSP.

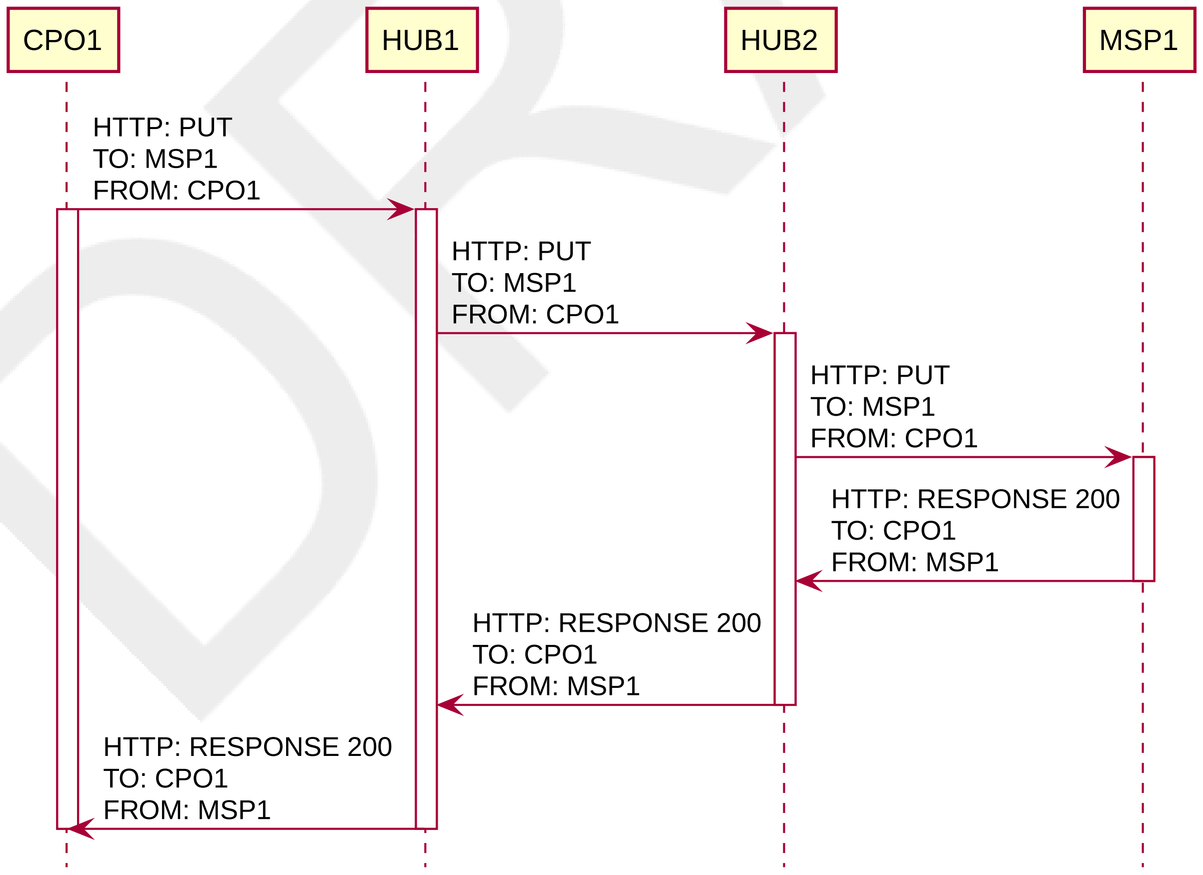

Figure 3. Example sequence diagram of a PUT via 2 Hubs.

3.1.7.1. Omitting from address in responses

When a party responses to a request (via a hub), and it was a request that contained the 'OCPI-to-' headers, the 'OCPI-from-' headers might be omitted. The requesting party knows to which party the request was send. It might be good practice to always provide the 'OCPI-from-' headers in a response, just to be complete and to give a requesting party to change to validate if the request was routed correctly by the Hub.

3.1.7.2. Broadcast push

For simplicity, connected clients might PUSH (POST, PUT, PATCH) information to all connected clients with an "opposite role", 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 send to Other, messages from "Other" are send to CPOs, not the eMSPs.)

Broadcast push might be very useful to push information like Locations or Tokens to all parties, connected to the Hub that have implemented the corresponding module. This means only 1 request to the Hub, not having to worry about the number of connected clients.

To send a Broadcast push, the client uses to Hubs party-id and country-code in the 'OCPI-to-' headers. The Hub parses the request and send a response to the client, with optionally its own party-id and country-code in the 'OCPI-from-' headers. The Hub then send to push to any client implementing the 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, to Open routing request should be used. (see below)

Broadcast push SHALL only be used with information that is meant to be send to all other parties. It is useful for things like: Tokens and Locations, but not for CDRs and Sessions.

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.

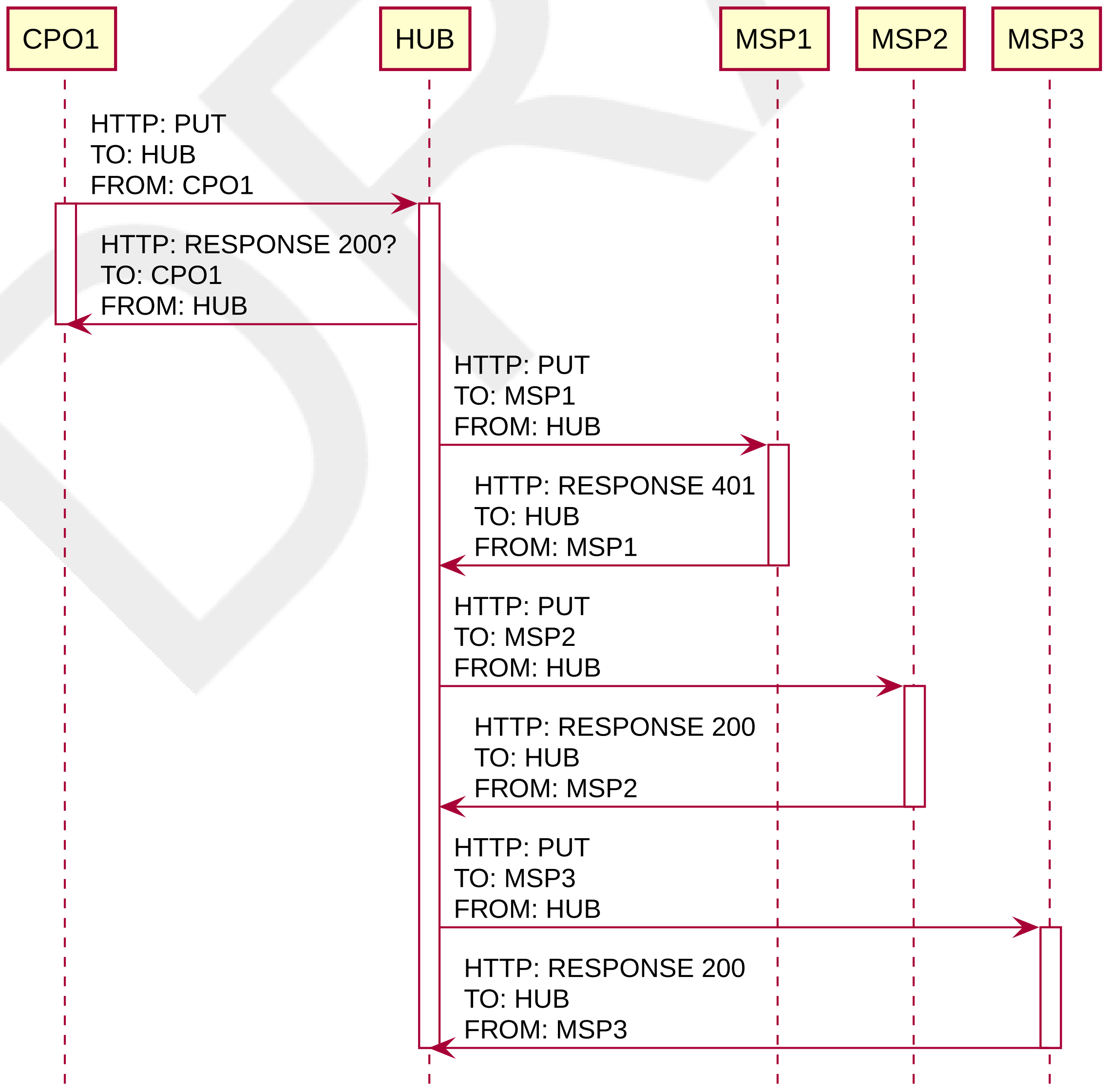

Figure 4. Example sequence diagram of a broadcast PUT from a CPO to multiple MSPs.

3.1.7.3. 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 a hub. The Hub can then decide to which part a request needs to be routed, or that it needs to be broadcasted.

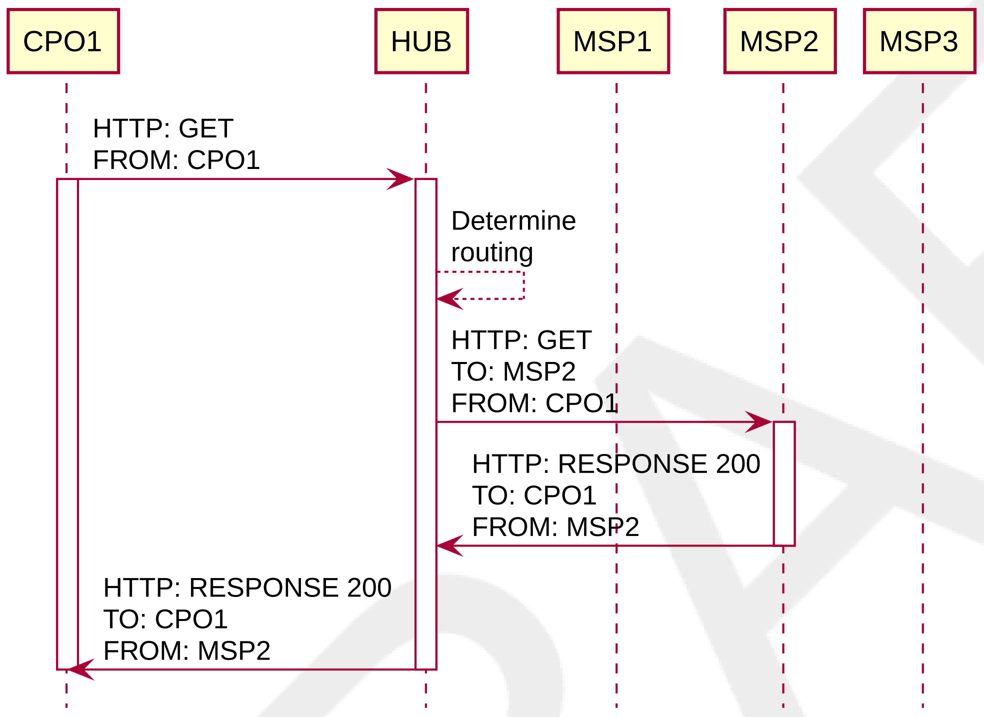

Figure 5. Example sequence diagram of a open routing GET from a CPO via the Hub.

GET All from Hub not possible

It is not possible for a client to ask the Hub for all objects within a certain module from all other parties.

Examples:

- A eMSP cannot ask the Hub for all the Locations the hub knows from all connected CPOs.
- A CPO cannot ask the Hub for all the Tokens a hub knows from all connected eMSPs.

The Tokens example: When the Hub would combine all the Tokens it knows and return them in one (paginated) response to the client, there will be Tokens from different MSPs in the same response, but there can only be one set of 'OCPI-from-' headers, which means the CPO cannot determine which Token belongs to which (virtual) MSP.

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

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

This is not an exclusive list, combinations are possible.

When the hub sends something to a virtual-party, the 'OCPI-to-' headers are required.

When the hub receives something from a virtual-party, the 'OCPI-from-' headers are required.

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 directly routed to a receiving party (and both parties are real CPO/eMSP, not virtual CPOs/eMSPs).

Name	Route	TO Header	FROM Header	Description
Direct request	Requesting party to Hub	Receiving-party		FROM headers are optional, Hub knows the requesting party
Direct request	Hub to receiving party		Requesting-party	TO headers are optional, party knows this request is for the party itself.
Direct response	receiving party to Hub	Requesting-party		FROM headers are optional, Hub knows the responding party
Direct response	Hub to requesting party		Receiving-party	TO headers are optional, party knows this response is for the party itself.

Virtual-party to Virtual-Party via Hub

This table contains the description of which header are required to be used for which message when a request is directly routed from a virtual party to a virtual party. In this scenario all headers are needed because the receiving/sending party is not the virtual-party themselves.

Name	Route	TO Header	FROM Header	Description
Direct request	Requesting party to Hub	Receiving-virtual-party	Requesting-virtual-party	
Direct request	Hub to receiving party	Receiving-virtual-party	Requesting-virtual-party	
Direct response	receiving party to Hub	Requesting-virtual-party	Receiving-virtual-party	
Direct response	Hub to requesting party	Requesting-virtual-party	Receiving-virtual-party	

Party to Party broadcast push

This table contains the description of which header are required to be used for which message when a request is a broadcast push to the hub.

Name	Route	TO Header	FROM Header	Description
Broadcast request	Requesting party to Hub	Hub		FROM headers are optional, Hub knows the requesting party
Broadcast response	Hub to requesting party		Hub	TO headers are optional, party knows this response is for the party itself.
Broadcast request	Hub to receiving party		Hub	TO headers are optional, party knows this request is for the party itself.
Broadcast response	receiving party to Hub	Hub		FROM headers are optional, Hub knows the responding party

Virtual-party to Virtual-Party broadcast push

This table contains the description of which header are required to be used for which message when a request is to be broadcasted by the hub and both parties are virtual parties. For a broadcast, the TO headers in the request from the requesting party to the Hub should contain the information of the Hub.

Name	Route	TO Header	FROM Header	Description
Broadcast request	Requesting party to Hub	Hub	Requesting-party	
Broadcast response	Hub to requesting party	Requesting-party	Hub	
Broadcast request	Hub to receiving party	Receiving-party	Hub	
Broadcast response	receiving party to Hub	Hub	Receiving-party	

Party to Party open routing request

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

Name	Route	TO Header	FROM Header	Description
Open request	Requesting party to Hub			Omitting the TO headers indicates to the Hub that the Hub has to figure out the routing.
Open request	Hub to receiving party		Requesting-party	TO headers can be omitted, when the receiving party is NOT a virtual party.
Open response	receiving party to Hub	Requesting-party	Receiving-party	FROM headers can be omitted, when the receiving party is NOT a virtual party.
Open response	Hub to requesting party	Requesting-party	Receiving-party	TO headers can be omitted, when the requesting party is NOT a virtual party.

3.2. 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 send via a Hub SHALL contain a unique correlation-ID, every response to this request SHALL contain the same ID.

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 the other party SHALL a new/unique value in the X-Request-ID HTTP header, not copy the X-Request-ID HTTP header from the incoming request that was 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)

TODO Add sequence diagram to show how X-Request-ID and X-Correlation-ID work together

3.3. 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/locations` and `/ocpi/emsp/2.2/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/credentials`
Charging location details	locations	`https://example.com/ocpi/cpo/2.2/locations`

eMSP interface	Identifier	Example URL
Credentials	credentials	`https://example.com/ocpi/emsp/2.2/credentials`
Charging location updates	locations	`https://example.com/ocpi/emsp/2.2/locations`

3.4. Offline behaviour

During communication over OCPI, it might happen that one of the communication parties is unreachable for an 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 requests fails or times out, the client should not queue the message and retry the same message again on a later time.

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 in-sync. For example: - CDRs of the period of communication loss can be rerieved with a GET command on the CDRs module, with filters to retrieve only CDRs of the period since the last CDR was received. - Status of EVSEs (or Locations) can be retrieved by calling a GET on the Locations module.

4. 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` is returned.

If a request is syntactically valid JSON and addresses an existing resource, no HTTP error should be returned. Those requests are supposed to have reached the OCPI layer. As is customary for RESTful APIs: if 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 is successfully updated.
- HTTP `201 - Created` when the object is 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.

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, `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.

4.1. 1xxx: Success

Code	Description
1000	Generic success code

4.2. 2xxx: Client errors

Errors detected by a server in the message sent by a client: The client did something wrong

Code	Description
2000	Generic client error
2001	Invalid or missing parameters
2002	Not enough information, for example: Authorization request with too little information.
2003	Unknown Location, for example: Command: START_SESSION with unknown location.

4.3. 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

Code	Description
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 minimum expected by the other party implementation.

4.4. 4xxx: Hub errors

When a server encounters an error, client side error (2xxx) or server side error (3xxx), it is send to status code to the Hub, the Hub SHALL forward this error to the client sending the request (when the request was not a broadcast push).

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

Code	Description
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)

5. *Versions* module

This is the required base module of OCPI. This is module is the start 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 version of OCPI.

5.1. 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 interface with your OCPI implementation, this endpoint is the starting point for discovering locations of the different modules and versions of OCPI that have been implemented.

Both the CPO and the eMSP must have this endpoint.

Method	Description
GET	Fetch information about the supported versions.

5.1.1. Data

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

5.1.2. Version *class*

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

5.1.3. GET

Fetch all supported OCPI versions of this CPO or eMSP.

5.1.3.1. Example

```
[
  {
    "version": "2.1.1",
    "url": "https://www.server.com/ocpi/2.1.1/"
  },
  {
    "version": "2.2",
    "url": "https://www.server.com/ocpi/2.2/"
  }
]
```

5.2. 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 are a Hub) can implement one version endpoint. With the information that is available in the version details, parties don't need to implement a separate endpoint per role (CPO or MSP) anymore. In the reality this means that when a company that is both a CPO and a MSP connects to another CPO/MSP combination, only one OCPI connection is needed.

NOTE OCPI 2.2 introduces 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`

`https://www.server.com/ocpi/emsp/2.2`

`https://ocpi.server.com/2.2/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 have this endpoint.

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

5.2.1. Data

Property	Type	Card.	Description
version	VersionNumber	1	The version number.
endpoints	Endpoint	+	A list of supported endpoints for this version.

5.2.2. Endpoint *class*

Property	Type	Card.	Description
identifier	ModuleID	1	Endpoint identifier.
role	InterfaceRole	1	Interface role this endpoint implements.

Property	Type	Card.	Description
url	URL	1	URL to the endpoint.

NOTE for the `credentials` module the role is not relevant as this module is the same for all roles.

5.2.3. InterfaceRole *enum*

Value	Description
CPO	CPO Interface implementation
MSP	MSP Interface implementation

5.2.4. ModuleID *enum*

The Module identifiers for each endpoint are 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 so that will be mentioned in the module description.

Module	ModuleID	Remark
CDRs	cdrs	
Charging Profiles	chargingprofiles	
Commands	commands	
Credentials & registration	credentials	Required for all implementations
Hub Client Info	hubclientinfo	
Locations	locations	
Sessions	sessions	
Tariffs	tariffs	
Tokens	tokens	

5.2.5. VersionNumber *enum*

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. (this version)

5.2.5.1. Custom Modules

Parties are allowed to create custom modules or customized versions of the existing modules. For this 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 (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`

5.2.6. GET

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

5.2.6.1. Examples

Simple version details example: CPO with only 2 modules.

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

Simple version details example: party with both CPO and MSP 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": "CPO",
      "url": "https://example.com/ocpi/2.2/credentials/"
    },
    {
      "identifier": "locations",
      "role": "CPO",
      "url": "https://example.com/ocpi/cpo/2.2/locations/"
    },
    {
      "identifier": "tokens",
      "role": "CPO",
      "url": "https://example.com/ocpi/cpo/2.2/locations/"
    },
    {
      "identifier": "locations",
      "role": "MSP",
      "url": "https://example.com/ocpi/msp/2.2/locations/"
    },
    {
      "identifier": "tokens",
      "role": "MSP",
      "url": "https://example.com/ocpi/msp/2.2/locations/"
    }
  ]
}
```

6. *Credentials* module

Module Identifier: `credentials`

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

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

6.1. Use cases

6.1.1. Registration

To register a CPO in an eMSP platform (or vice versa), the CPO must create a unique credentials token that can be used for authenticating the eMSP. This credentials token along with the versions endpoint should be sent to the eMSP in a secure way that is outside the scope of this protocol.

`CREDENTIALS_TOKEN_A` is given offline, after registration store the `CREDENTIALS_TOKEN_C` which will be used in future exchanges. The `CREDENTIALS_TOKEN_A` can then be thrown away.

The eMSP starts the registration process, retrieves the version information and details (using `CREDENTIALS_TOKEN_A` in the HTTP Authorization header). The eMSP generates `CREDENTIALS_TOKEN_B`, send it to the CPO in a POST request to the `credentials` module of the CPO. The CPO stores `CREDENTIALS_TOKEN_B` and uses it for any requests to the eMSP, including the version information and details.

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

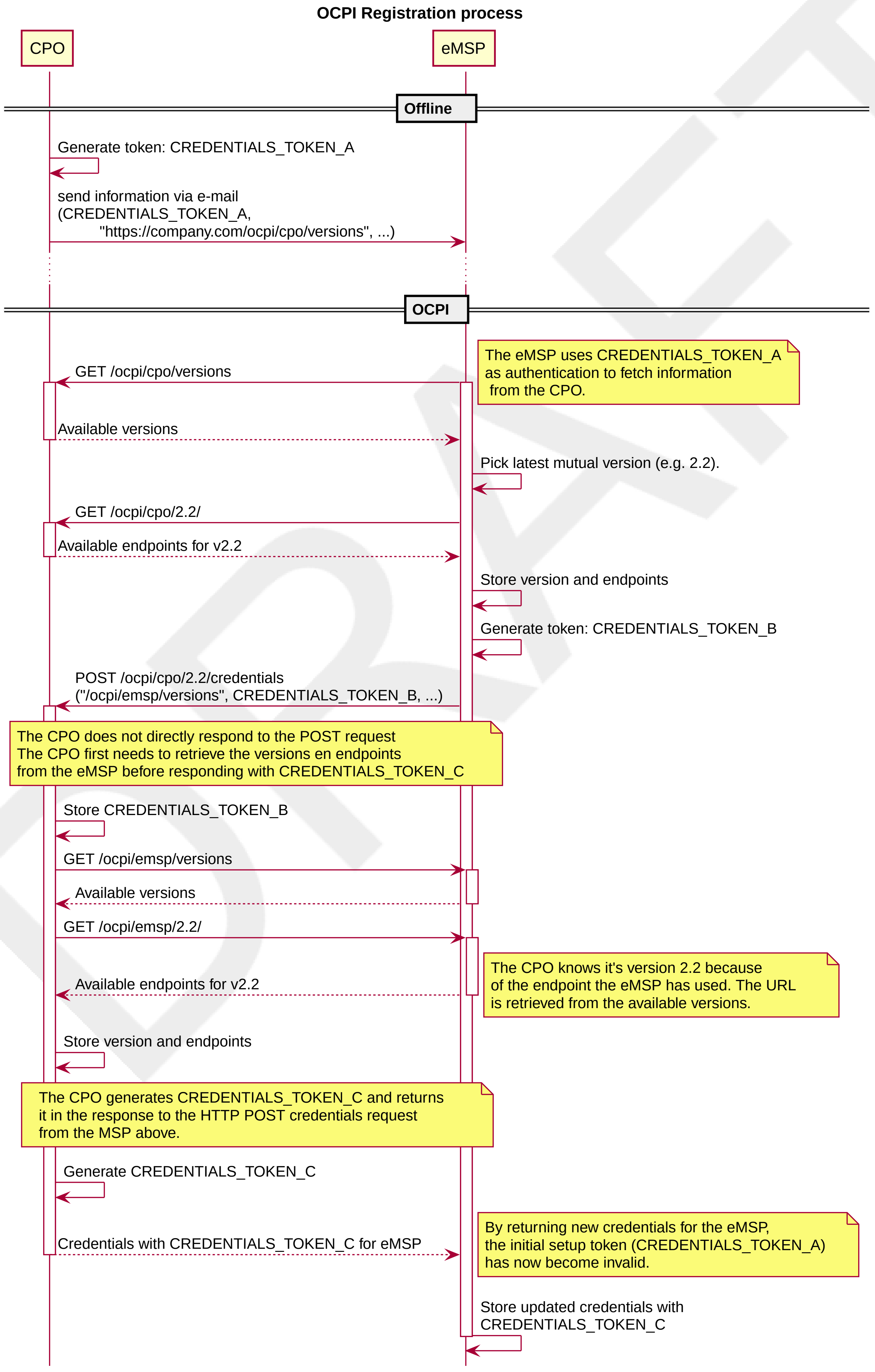
OCPI Registration process
CPO
eMSP
Offline
Generate token: CREDENTIALS_TOKEN_A
send information via e-mail (CREDENTIALS_TOKEN_A, "https://company.com/ocpi/cpo/versions", ...)
OCPI
The eMSP uses CREDENTIALS_TOKEN_A as authentication to fetch information from the CPO.
GET /ocpi/cpo/versions
Available versions
Pick latest mutual version (e.g. 2.2).
GET /ocpi/cpo/2.2/
Available endpoints for v2.2
Store version and endpoints
Generate token: CREDENTIALS_TOKEN_B
POST /ocpi/cpo/2.2/credentials ("/ocpi/emsp/versions", CREDENTIALS_TOKEN_B, ...)
The CPO does not directly respond to the POST request The CPO first needs to retrieve the versions en endpoints from the eMSP before responding with CREDENTIALS_TOKEN_C
Store CREDENTIALS_TOKEN_B
GET /ocpi/emsp/versions
Available versions
GET /ocpi/emsp/2.2/
The CPO knows it's version 2.2 because of the endpoint the eMSP has used. The URL is retrieved from the available versions.
Available endpoints for v2.2
Store version and endpoints
The CPO generates CREDENTIALS_TOKEN_C and returns it in the response to the HTTP POST credentials request from the MSP above.
Generate CREDENTIALS_TOKEN_C
By returning new credentials for the eMSP, the initial setup token (CREDENTIALS_TOKEN_A) has now become invalid.
Credentials with CREDENTIALS_TOKEN_C for eMSP
Store updated credentials with CREDENTIALS_TOKEN_C

Figure 6. The OCPI registration process

Due to its symmetric nature, the CPO and eMSP can be swapped in the registration sequence.

6.1.2. Updating to a newer version

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

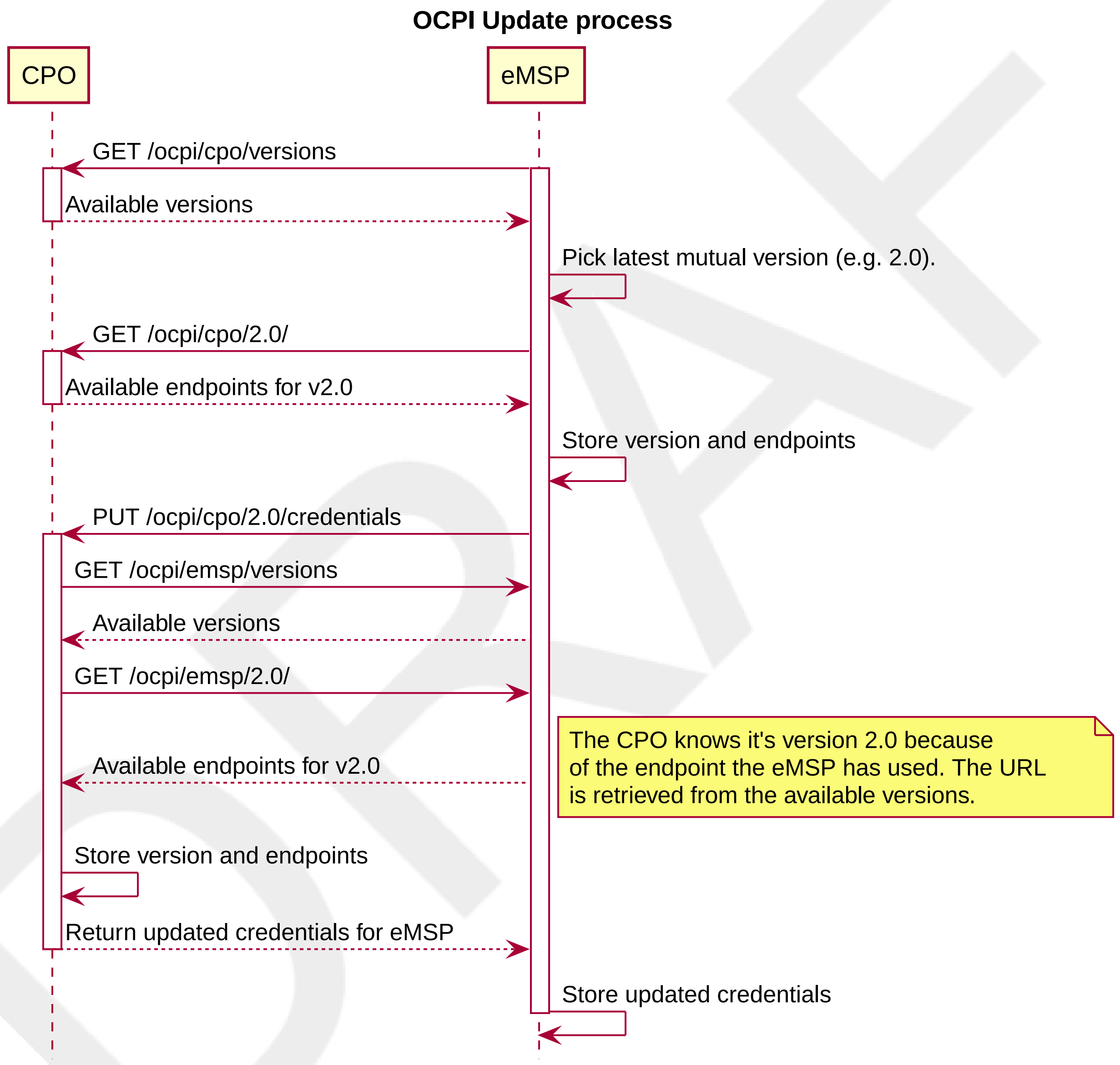
OCPI Update process
CPO
eMSP
GET /ocpi/cpo/versions
Available versions
Pick latest mutual version (e.g. 2.0).
GET /ocpi/cpo/2.0/
Available endpoints for v2.0
Store version and endpoints
PUT /ocpi/cpo/2.0/credentials
GET /ocpi/emsp/versions
Available versions
GET /ocpi/emsp/2.0/
The CPO knows it's version 2.0 because of the endpoint the eMSP has used. The URL is retrieved from the available versions.
Available endpoints for v2.0
Store version and endpoints
Return updated credentials for eMSP
Store updated credentials

Figure 7. The OCPI update process

6.1.3. 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 party will fetch and store the corresponding set of endpoints.

6.1.4. 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.

6.1.5. 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.

6.1.6. Required endpoints not available

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

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

In case the client is starting the credentials exchange process and cannot find the endpoints it expects, it is expected NOT to send the POST request with credentials to the server. Log a message/notify the administrator to contact the administrator of the server system.

In case the server, receiving the request from a client, cannot find the endpoints it expects, then it is expected to respond to the request with a status code: 3003.

6.2. Interfaces and endpoints

Example: `/ocpi/cpo/2.2/credentials` and `/ocpi/emsp/2.2/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).

6.2.1. 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.

6.2.2. 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 must return a `HTTP status code 405: method not allowed` if the client was already registered.

6.2.3. 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 must return a `HTTP status code 405: method not allowed` if the client was not registered yet.

6.2.4. 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 must return a `HTTP status code 405: method not allowed` if the client was not registered.

6.3. Object description

6.3.1. Credentials object

Property	Type	Card.	Description
token	string(64)	1	The credentials token for the other party to authenticate in your system.
url	URL	1	The URL to your API versions endpoint.
roles	CredentialsRole	+	List of the roles this party provides.

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

A party can have the some role more then once, 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 determine the URLs a client will use when pushing a client owned object. The `country_code` is added the make certain the URL used when pushing a client owned object is unique, there might be multiple parties in the world with the same `party_id`, but the combination should always be unique. A party operating in multiple countries can always use the home country of the company for all connections.

For example: an OCPI implementation might push EVSE IDs from a company for different countries, preventing an OCPI connection per country a company is operating in.

The `party_id` and `country_code` give here, have no direct link with the eMI3 EVSE IDs and Contract IDs that might be used in the different OCPI modules.

For example: an party implementing OCPI might push EVSE IDs with an eMI3 `spot operator` different from the OCPI `party_id` and/or the `country_code`.

6.3.2. 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/MSP 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": "MSP",
        "party_id": "EXA",
        "country_code": "NL",
        "business_details": {
            "name": "Example Provider"
        }
    }]
}
```

Example of 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 CPO credentials object with virtual CPOs

```
{
    "token": "9e80aca8-28be-11e9-b210-d663bd873d93",
    "url": "https://ocpi.example.com/versions/",
    "roles": [{
        "role": "CPO",
        "party_id": "EXA",
        "country_code": "NL",
        "business_details": {
            "name": "Operator"
        }
    }, {
        "role": "CPO",
        "party_id": "VPA",
        "country_code": "NL",
        "business_details": {
            "name": "Virtual Provider 1"
        }
    }, {
        "role": "CPO",
        "party_id": "VPB",
        "country_code": "NL",
        "business_details": {
            "name": "Virtual Provider 2"
        }
    }]
}
```

6.4. Data types

6.4.1. 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 15118 ISO standard).
country_code	CiString(2)	1	Country code of the country this party is operating in.

6.4.2. Role *enum*

Value	Description
CPO	Charge Point Operator Role
MSP	eMobility Service Provider Role
HUB	Hub role
NSP	Navigation Service Provider Role
OTHER	Other role
SCSP	Smart Charging Service Provider Role

7. *Locations* module

Module Identifier: `locations`

Data owner: `CPO`

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

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

7.1. Flow and Lifecycle

The Locations module has Locations as base object, Locations have EVSEs, EVSEs have Connectors. With the methods in the eMSP interface, Location information/statuses can be shared with the eMSP. Updates can be done to the Location, but also to only an EVSE or a Connector.

When a CPO creates Location objects it pushes them to the eMSPs by calling PUT on the eMSPs Locations endpoint. Providers who do not support push mode need to call GET on the CPOs Locations endpoint to receive the new object.

If the CPO wants to replace a Location related object, they 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 the eMSP by calling the PATCH on the eMSPs Locations endpoint. 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 they must update by setting the `status` field to `REMOVED` and call the PUT or PATCH on the eMSP system. A *Location* without valid *EVSE* objects can be considered as expired and should no longer be displayed. There is no direct way to delete a location, EVSE or Connector, there are other modules like `sessions` that link to location, EVSE and Connector IDs. If they were removed, these 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 eMSPs system, the CPO can call the GET to validate the object in the eMSP system.

7.2. Interfaces and endpoints

There is both a CPO and an eMSP interface for Locations. Advised is to use the push direction from CPO to eMSP during normal operation. The CPO interface is meant to be used when the connection between 2 parties is established, to retrieve the current list of Location objects with the current status, and when the eMSP is not 100% sure the Locations cache is completely correct. The eMSP can use the CPO GET Object interface to retrieve a specific Location, EVSE or Connector, this might be used by a eMSP that wants information about a specific Location, but has not implemented the eMSP Locations interface (cannot receive push).

7.2.1. CPO Interface

Method	Description
GET	Fetch a list 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

7.2.1.1. GET Method

Depending on the URL Segments provided, the GET request can either be used to retrieve information about a list of available locations and EVSEs at this CPO: GET List Or it can be used to get information about a 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={l
imit}]
```

Examples:

```
https://www.server.com/ocpi/cpo/2.2/locations/?date_from=2019-01-28T12:00:00&date_to=2019-01-
29T12:00:00
```

```
https://ocpi.server.com/2.2/locations/?offset=50
```

```
https://www.server.com/ocpi/2.2/locations/?date_from=2019-01-29T12:00:00&limit=100
```

```
https://www.server.com/ocpi/cpo/2.2/locations/?offset=50&amp;limit=100
```

If additional parameters: {date_from} and/or {date_to} are provided, only Locations with (`last_updated`) between the given date_from and date_to will be returned. If an EVSE is updated, also the 'parent' Location's `last_updated` fields is updated. If a Connector is updated, the EVSE's `last_updated` and the Location's `last_updated` field are 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 this Date/Time.
date_to	DateTime	no	Only return Locations that have `last_updated` before this Date/Time.
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

The endpoint returns a list of Location objects 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
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/locations/LOC1
```

```
https://www.server.com/ocpi/cpo/2.2/locations/LOC1/3256
```

```
https://www.server.com/ocpi/cpo/2.2/locations/LOC1/3256/1
```

The following parameters can be provided as URL segments.

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.
> Location	1	If an EVSE object was requested: the EVSE object.
> Connector	1	If a Connector object was requested: the Connector object.

7.2.1.2. eMSP Interface

Locations is a client owned object, 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/locations/BE/BEC/LOC1
```

```
https://server.com/ocpi/2.2/locations/BE/BEC/LOC1/3256
```

```
https://ocpi.server.com/2.2/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 Connectors to the eMSP
PATCH	Notify the eMSP of partial updates to a Location, EVSEs or Connector (such as the status).
DELETE	n/a *(use PATCH)*

7.2.1.3. 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 this PUT to the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to 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.
> Location	1	If an EVSE object was requested: the EVSE object.
> Connector	1	If a Connector object was requested: the Connector object.

7.2.1.4. PUT Method

The CPO pushes available Location/EVSE or Connector objects to the eMSP. PUT is used to send new Location objects to the eMSP, or to replace existing Locations.

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.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the CPO requesting this PUT to the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
location_id	CiString(36)	yes	Location.id of the new Location object, or the Location of which an EVSE or Location object is send
evse_uid	CiString(36)	no	Evse.uid, required when an EVSE or Connector object is send/replaced.
connector_id	CiString(36)	no	Connector.id, required when a Connector object is send/replaced.

Request Body

The request contains the new/updated object.

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

7.2.1.5. 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.

Example: a simple status update

This is the most common type of update message to notify eMSPs that an EVSE (EVSE with uid 3255 of Charge Point 1012) is now occupied.

```
PATCH To URL: https://www.server.com/ocpi/emsp/2.2/locations/NL/TNM/1012/3255

{
  "status": "CHARGING"
}
```

Example: change the location name

In this example the name of location 1012 is updated.

```
PATCH To URL: https://www.server.com/ocpi/emsp/2.2/locations/NL/TNM/1012


{
  "name": "Interparking Gent Zuid"
}
```

Example: set tariff update

In this example connector 2 of EVSE 1 of Charge Point 1012, receives a new pricing scheme.

```
PATCH To URL: https://www.server.com/ocpi/emsp/2.2/locations/NL/TNM/1012/3255/2

{
  "tariff_id": "15"
}
```

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 is new, 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/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_id": "14"
    }
  ],
  "physical_reference": 3,
  "floor": -1
}
```

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/locations/NL/TNM/1012/3256

{
  "status": "REMOVED"
}
```

Note: To inform that an EVSE is scheduled for removal, the status_schedule field can be used.

7.3. Object description

Location, EVSE and Connector have the following relation.

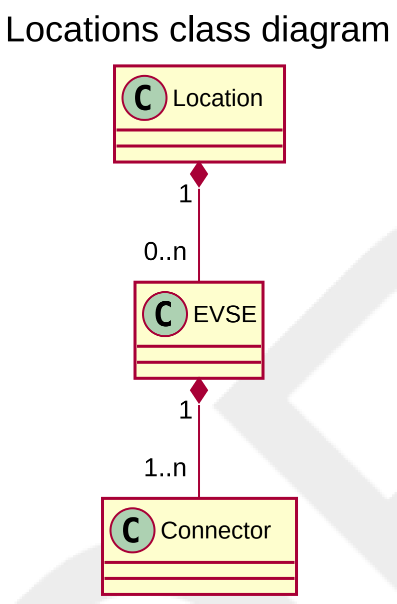

Figure 8. Location class diagram

7.3.1. *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.

Property	Type	Card.	Description
id	CiString(36)	1	Uniquely identifies the location within the CPOs platform (and suboperator platforms). This field can never be changed, modified or renamed.
type	LocationType	1	The general type of the charge point 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 location at highways don’t have postal codes.
state	string(20)	?	State or Province of the location, only use when relevant.
country	string(3)	1	ISO 3166-1 alpha-3 code for the country of this location.

Property	Type	Card.	Description
coordinates	GeoLocation	1	Coordinates of the location.
related_locations	AdditionalGeoLocation	*	Geographical location of related points relevant to the user.
evses	Location	*	List of EVSEs that belong to 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 should be used instead.
suboperator	BusinessDetails	?	Information of the suboperator if available.
owner	BusinessDetails	?	Information of the owner if available.
facilities	Facility	*	Optional list of facilities this charge location directly belongs to.
time_zone	string(255)	?	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.
last_updated	DateTime	1	Timestamp when this Location or one of its EVSEs or Connectors were last updated (or created).

7.3.1.1. Example

```
{
  "id": "LOC1",
  "type": "ON_STREET",
  "name": "Gent Zuid",
  "address": "F.Rooseveltlaan 3A",
  "city": "Gent",
  "postal_code": "9000",
  "country": "BEL",
  "coordinates": {
    "latitude": "51.047599",
    "longitude": "3.729944"
  },
  "evses": [{
    "uid": "3256",
    "evse_id": "BE*BEC*E041503001",
    "status": "AVAILABLE",
    "status_schedule": [],
    "capabilities": [
      "RESERVABLE"
    ],
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "CABLE",
      "power_type": "AC_3_PHASE",
      "voltage": 220,
      "amperage": 16,
      "tariff_id": "11",
      "last_updated": "2015-03-16T10:10:02Z"
    }, {
      "id": "2",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "voltage": 220,
      "amperage": 16,
      "tariff_id": "11",
      "last_updated": "2015-03-18T08:12:01Z"
    }],
    "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",
      "voltage": 220,
      "amperage": 16,
      "tariff_id": "12",
      "last_updated": "2015-06-29T20:39:09Z"
    }],
    "physical_reference": "2",
    "floor_level": "-2",
    "last_updated": "2015-06-29T20:39:09Z"
  }],
  "operator": {
    "name": "BeCharged"
  },
  "last_updated": "2015-06-29T20:39:09Z"
}
```

7.3.2. *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. It will only contain directions to get from the location to the EVSE (i.e. *floor*, *physical_reference* or *directions*). When these properties are insufficient to reach the EVSE from the *Location* point, then it typically indicates that this 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). For example a database unique ID or the "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. This field is named `uid` instead of `id`, because `id` could be confused with `evse_id` which is a eMI3 defined field.
evse_id	CiString(48)	?	Compliant with the following specification for EVSE ID from "eMI3 standard version V1.0" (http://emi3group.com/documents-links/) "Part 2: business objects." Optional because: if an EVSE ID is to be re-used the EVSE ID can be removed from an EVSE that is removed (status: REMOVED)
status	Status	1	Indicates the current status of the EVSE.
status_schedule	StatusSchedule	*	Indicates a planned status in the future 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 charging station 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.
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	*	The restrictions that apply to the parking spot.
images	Image	*	Links to images related to the EVSE such as photos or logos.
last_updated	DateTime	1	Timestamp when this EVSE or one of its Connectors was last updated (or created).

7.3.3. *Connector* Object

A connector is the socket or cable 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	
voltage	int	1	Voltage of the connector (line to neutral for AC_3_PHASE), in volt [V].
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 watt [W]. When the maximum electric power is lower then the calculated value from: `voltage` and `amperage`, this value should be set. For example: A Charge Point which can deliver up to 920V, up to 400A, but max. 150kW. Depending on the car, it may supply max voltage or current, but not both.
tariff_ids	CiString(36)	*	Identifiers of the current valid charging tariffs. Multiple tariffs are possible, but each require a unique Tariff.type, duplicates not allowed. When Preference based Smart Charging is supported, on tariff for every possible ProfileType should be provided, these tell the Driver the options he has 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.
last_updated	DateTime	1	Timestamp when this Connectors was last updated (or created).

7.4. Data types

7.4.1. AdditionalGeoLocation *class*

This class defines a geo location. 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.

7.4.2. BusinessDetails *class*

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

7.4.3. Capability *enum*

The capabilities of an EVSE.

Value	Description
CHARGING_PROFILE_CAPABLE	The EVSE supports charging profiles.
CHARGING_PREFERENCES_CAPABLE	The EVSE supports charging preferences.
CREDIT_CARD_PAYABLE	Charging at this EVSE can be payed with a credit card.
DEBIT_CARD_PAYABLE	Charging at this EVSE can be payed with a debit card.
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 a RFID token

Value	Description
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.

7.4.4. 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.

7.4.5. ConnectorType *enum*

The socket or plug standard of the charging point.

Value	Description
CHADEMO	The connector type is CHAdeMO, 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
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 phase 16 Amperes (usually red)
IEC_60309_2_three_32	IEC 60309-2 Industrial Connector three phase 32 Amperes (usually red)
IEC_60309_2_three_64	IEC 60309-2 Industrial Connector three phase 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"
TESLA_R	Tesla Connector "Roadster"-type (round, 4 pin)
TESLA_S	Tesla Connector "Model-S"-type (oval, 5 pin)

7.4.6. 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.)*

7.4.6.1. 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": [
  { "source": "NUCLEAR_WASTE",  "amount": 0.0006   },
  { "source": "CARBON_DIOXIDE", "amount": 372      }
  ],
  "supplier_name":       "E.ON Energy Deutschland",
  "energy_product_name": "E.ON DirektStrom eco"
}
```

7.4.7. EnergySource *class*

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

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

7.4.8. 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.

7.4.9. 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.

7.4.10. EnvironmentalImpactCategory *enum*

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.

7.4.11. ExceptionalPeriod *class*

Specifies one exceptional period for opening or access hours.

Property	Type	Card.	Description
period_begin	DateTime	1	Begin of the exception.
period_end	DateTime	1	End of the exception.

7.4.12. Facility *enum*

Value	Description
HOTEL	A hotel.

Value	Description
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/park etc.
MUSEUM	A museum.
BUS_STOP	A bus stop.
TAXI_STAND	A taxi stand.
TRAIN_STATION	A train station.
AIRPORT	An airport.
CARPOOL_PARKING	A carpool parking.
FUEL_STATION	A Fuel station.
WIFI	Wifi or other type of internet available.

7.4.13. GeoLocation *class*

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 minimal for GPS coordinates for Charging Stations, this gives approximately 1 meter precision. More is always better. Seven decimal places gives approximately 1cm precision.

7.4.14. 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`.
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 regularHours and exceptionalOpenings. Should not overlap exceptionalOpenings.

7.4.15. Image *class*

This class references images related to a 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 OCHP. 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 they should be displayed additionally, not optionally.

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

Logo Dimensions: The recommended dimensions for logos are exactly 512 pixels wide and 512 pixels height. Thumbnail representations for 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	string(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

7.4.16. ImageCategory *enum*

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 view
OPERATOR	logo of the charge points operator, for example a municipality, to be displayed with the EVSEs detailed information view or in lists and maps, if no networkLogo is present
OTHER	Other
OWNER	logo of the charge points owner, for example a local store, to be displayed with the EVSEs detailed information view

7.4.17. LocationType *enum*

Reflects the general type of the charge points location. May be used for user information.

Value	Description
ON_STREET	Parking in public space.
PARKING_GARAGE	Multistorey car park.
UNDERGROUND_GARAGE	Multistorey car park, mainly underground.
PARKING_LOT	A cleared area that is intended for parking vehicles, i.e. at super markets, bars, etc.
OTHER	None of the given possibilities.
UNKNOWN	Parking location type is not known by the operator (default).

7.4.18. ParkingRestriction *enum*

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

Value	Description
EV_ONLY	Reserved parking spot for electric vehicles.
PLUGGED	Parking is only allowed while plugged in (charging).
DISABLED	Reserved parking spot for disabled people with valid ID.
CUSTOMERS	Parking spot for customers/guests only, for example in case of a hotel or shop.
MOTORCYCLES	Parking spot only suitable for (electric) motorcycles or scooters.

7.4.19. PowerType *enum*

Value	Description
AC_1_PHASE	AC mono phase.
AC_3_PHASE	AC 3 phase.
DC	Direct Current.

7.4.20. 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 given in hours and minutes. Must be in 24h format with leading zeros. Example: "18:15". Hour/Minute separator: ":" Regex: [0-2][0-9]:[0-5][0-9]
period_end	string(5)	1	End of the regular period, syntax as for period_begin. Must be later than period_begin.

7.4.20.1. 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	-	-

7.4.21. 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 it is no longer available (deleted).

Value	Description
OUTOFORDER	The EVSE/Connector is currently out of order.
PLANNED	The EVSE/Connector is planned, will be operating soon
REMOVED	The EVSE/Connector/charge point is 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)

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

8. *Sessions* module

Module Identifier: `sessions`

Data owner: `CPO`

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.

8.1. Flow and Lifecycle

8.1.1. Push model

When the CPO creates a Session object they push it to the eMSPs by calling PUT on the eMSPs 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 eMSPs 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 eMSPs system, the CPO can call the GET to validate the Session object in the eMSP system.

8.1.2. Pull model

eMSPs who do not support the push model need to call GET on the CPOs Sessions endpoint to receive a list of Sessions.

This GET can also be used, combined with the Push model to retrieve Sessions after the system (re)connects to a CPO, to get a list Sessions 'missed' during a time offline.

8.1.3. 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, need to know how much energy an EV needs before what time. Via a PUT on the CPO Interface, during an ongoing session, the eMSP can send Charging Preferences for the driver. Indicating the preferences of the driver to the CPO.

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 preferences different prices. A Connector can have multiple Tariffs one for each ProfileType.

8.2. Interfaces and endpoints

8.2.1. CPO Interface

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 on an ongoing session.
PATCH	n/a

Method	Description
DELETE	n/a

8.2.1.1. GET Method

Fetch Sessions from the CPO systems.

Endpoint structure definition:

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

Examples:

```
https://www.server.com/ocpi/cpo/2.2/sessions/?date_from=2019-01-28T12:00:00&date_to=2019-01-
29T12:00:00
```

```
https://ocpi.server.com/2.2/sessions/?offset=50
```

```
https://www.server.com/ocpi/2.2/sessions/?date_from=2019-01-29T12:00:00&limit=100
```

```
https://www.server.com/ocpi/cpo/2.2/sessions/?offset=50&amp;limit=100
```

Request Parameters

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

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

Parameter	Datatype	Required	Description
date_from	DateTime	yes	Only return Sessions that have `last_updated` after this Date/Time.
date_to	DateTime	no	Only return Sessions that have `last_updated` before this Date/Time.
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 as 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

8.2.1.2. PUT Method

Set/update the drivers 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/sessions/1234/charging_preferences
```

The `/charging_preferences` 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 on which these 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 for the driver for this Session.

Response Data

The endpoint response contains a ChargingPreferencesResponse value.

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

8.2.1.3. eMSP Interface

Sessions is a client owned object, so the end-points 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/cpo/2.2/sessions/BE/BEC/1234
```

Method	Description
GET	Get the Session object from the eMSP system by its id {session_id}.
POST	n/a
PUT	Send a new/updated Session object
PATCH	Update the Session object of id {session_id}.
DELETE	n/a

8.2.1.4. GET Method

The CPO system might request the current version of a Session object from the eMSP system for, for example validation purposes, or the CPO system might have received a error on a PATCH.

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 this GET to the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this GET to the eMSP system.
session_id	CiString(36)	yes	id of the Session object to get from the eMSP system.

Response Data

The response contains the request Session object, if available.

Datatype	Card.	Description
Session	1	Session object requested.

8.2.1.5. PUT Method

Inform the system about a new/updated session in the eMSP backoffice by PUTing a *Session* object.

Request Body

The request contains the new or updated Session object.

Type	Card.	Description
Session	1	new Session object.

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 this PUT to the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
session_id	CiString(36)	yes	id of the new or updated Session object.

8.2.1.6. 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.

Example: update the total cost

```
PATCH To URL: https://www.server.com/ocpi/cpo/2.2/sessions/NL/TNM/101

{
  "total_cost": {
    "excl_vat": 0.60,
    "incl_vat": 0.66
  }
}
```

8.3. Object description

8.3.1. *Session* Object

Property	Type	Card.	Description
id	CiString(36)	1	The unique id that identifies the session in the CPO platform.
start_datetime	DateTime	1	The time when the session became active.
end_datetime	DateTime	?	The time when the session is completed.
kwh	number	1	How many kWh are charged.
cdr_token	CdrToken	1	Token used to start this Charging Session, includes all the relevant information to identify the unique token.
auth_method	AuthMethod	1	Method used for authentication.
location_id	CiString(36)	1	Location.id of the Location object of this CPO, on which the Charging Session is ongoing.
evse_uid	CiString(36)	1	EVSE.uid of the EVSE of this Location on which the Charging Session is ongoing.
connector_id	CiString(36)	1	Connector.id of the Connector of this Location the Charging Session is ongoing.
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 (excluding VAT) 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, no price information is given in the Session object, this does not have to mean it is free of charge.
status	SessionStatus	1	The status of the session.
last_updated	DateTime	1	Timestamp when this Session was last updated (or created).

8.3.1.1. Examples

Simple Session example of a just starting session

```
{
  "id": "101",
  "start_datetime": "2015-06-29T22:39:09Z",
  "kwh": 0.00,
  "token_uid": "012345678",
  "token_type": "RFID",
  "auth_method": "WHITELIST",
  "location_id": "LOC1",
  "evse_uid": "3256",
  "connector_id": "1",
  "currency": "EUR",
  "total_cost": {
    "excl_vat": 2.50,
    "incl_vat": 2.75
  },
  "status": "PENDING",
  "last_updated": "2015-06-29T22:39:09Z"
}
```

Simple Session example of a short finished session

```
{
  "id": "101",
  "start_datetime": "2015-06-29T22:39:09Z",
  "end_datetime": "2015-06-29T23:50:16Z",
  "kwh": 41.00,
  "token_uid": "012345678",
  "token_type": "RFID",
  "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": {
    "excl_vat": 8.50,
    "incl_vat": 9.35
  },
  "status": "COMPLETED",
  "last_updated": "2015-06-29T23:50:17Z"
}
```

8.3.2. *ChargingPreferences* Object

Contains the charging preferences for 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, for every supported ProfileType, a Tariff is provided. This gives the EV Driver the option between different pricing options.
departure_time	DateTime	?	Expected departure. Driver has given this datetime as expected departure moment, which does not mean that that will be the real departure time.
energy_need	number	?	Requested amount of energy in kWh. EV drivers wants to have this amount of energy charged.
discharge_allowed	boolean	?	Driver allows his/her EV to be discharged when needed, as long as the other preferences are met: EV is charged with the preferred energy at the preferred momement. Default if omitted: **false**.

8.4. Data types

Describe all datatypes used in this object

8.4.1. ChargingPreferencesResponse *enum*

Different smart charging profile types.

No value for NOT_SUPPORTED, when a PUT for ChargingPreferences is received for a EVSE that does not support it: use HTTP 404.

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

8.4.2. 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 which as much regenerative (green) power as possible.
REGULAR	Driver does not have special preferences.

8.4.3. SessionStatus *enum*

Defines the state of a session.

Value	Description
ACTIVE	The session is accepted and active. Al pre-condition are 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 is finished successfully. No more modifications will be made to this session.
INVALID	The session is declared invalid and will not be billed.
PENDING	The session is pending, it has not yet started. Not all pre-condition are met. This is the initial state. This session might never become an *active* session.

9. *CDRs* module

Module Identifier: `cdrs`

Data owner: `CPO`

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. There is no requirement to send CDRs 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.

9.1. Flow and Lifecycle

CDRs are created by the CPO. They probably only will be sent to the eMSP that will be paying the bill of a charging session. Because a CDR is for billing purposes, it cannot be changed/replaced, once sent to the eMSP, changes are not allowed in a CDR, a Credit CDR needs to be send.

9.1.1. Credit CDRs

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

When a CPO wants to make changes to a CDR that is already send the a CPO, the CPO has to send a credit CDR for the first CDR. This credit CDR SHALL have a different CDR.id, this 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 this is a Credit CDR, the `credit` field is 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 send the Credit CDR, the CPO can send a new CDR, with a new unique ID, with 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, that is up the business contracts between the different parties involved, there might be local laws involved etc.

9.1.2. 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 costs. It is advised to send enough information to the eMSP so it can calculate its own costs for billing their customer. An eMSP might have a very different contract/pricing model with the EV driver than the tariff structure from the CPO.

NOTE: CDRs can not yet be updated or removed. This might be added in a future version of OCPI.

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

9.1.3. Pull model

eMSPs who do not support the push model need to call GET on the CPOs CDRs endpoint to receive a list of CDRs.

This GET can also be used, combined with the Push model to retrieve CDRs, after the system (re)connects to a CPO, to get a list of CDRs, 'missed' during a time offline.

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.

9.2. Interfaces and endpoints

There is both a CPO and an eMSP interface for CDRs. Depending on business requirements parties can decide to use the CPO Interface/Get model, or the eMSP Interface/Push model, or both. Push is the preferred model to use, the eMSP will receive CDRs when created by the CPO.

9.2.1. CPO Interface

The CDRs endpoint can be used to create or 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/cdrs/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00
```

```
https://ocpi.server.com/2.2/cdrs/?offset=50
```

```
https://www.server.com/ocpi/2.2/cdrs/?date_from=2019-01-29T12:00:00&limit=100
```

```
https://www.server.com/ocpi/cpo/2.2/cdrs/?offset=50&amp;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

9.2.1.1. GET Method

Fetch CDRs from the CPO systems.

Request Parameters

If additional parameters: {date_from} and/or {date_to} are provided, only CDRs with `last_updated` between the given date_from and date_to 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 this Date/Time.
date_to	DateTime	no	Only return CDRs that have `last_updated` before this Date/Time.
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 as 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.

9.2.2. eMSP Interface

The CDRs endpoint can be used to create, or get 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)

9.2.2.1. GET Method

Fetch CDRs from the eMSP system.

Endpoint structure definition:

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

Example:

```
https://www.server.com/ocpi/2.2/cdrs/1234
```

Response URL

To retrieve an existing URL from the eMSP 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.

9.2.2.2. POST Method

Creates a new CDR.

The post method should contain the full, final CDR object.

Endpoint structure definition:

```
{cdr_endpoint_url}
```

Example:

```
https://www.server.com/ocpi/2.2/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 system.

Parameter	Datatype	Required	Description
Location	URL	yes	URL to the newly created CDR in the eMSP system, can be used by the CPO system to do a GET on of the same CDR

The MSP 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/cdrs/123456
```

9.3. Object description

9.3.1. *CDR* Object

The *CDR* object describes the Charging Session and its costs, how these costs are built up, 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 CDR will be recognized as correct by the driver and not contested.

The *CDR* object shall always contain information like location, EVSE, Tariffs and Token as they were at the START of the Charging Session.

Property	Type	Card.	Description
id	CiString(39)	1	Uniquely identifies the CDR within the CPOs platform (and sub-operator platforms). This field is longer then the usual 36 chars to allow for credit CDRs to have something appended to the original ID, to make it unique. 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.
stop_date_time	DateTime	1	Stop timestamp of the charging session.
session_id	CiString(36)	?	Unique ID of the Session for which this CDR is send. Is only allowed to be omitted when the CPO has not implemented the Sessions Module.

Property	Type	Card.	Description
cdr_token	CdrToken	1	Token used to start this Charging Session, includes all the relevant information to identify the unique token.
auth_method	AuthMethod	1	Method used for authentication.
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 tariff elements, 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. A session consists of 1 or more periods, where each period has a different relevant Tariff.
total_cost	Price	1	Total cost of this transaction in the specified currency.
total_energy	number	1	Total energy charged, in kWh.
total_time	number	1	total duration of this session (including the duration of charging and not charging), in hours.
total_parking_time	number	?	Total duration during this session that the EV is not being charged (no energy being transfered between EVSE and EV), in hours.
remark	string(255)	?	Optional remark, can be used to provide addition human readable information to the CDR, for example: reason why a transaction was stopped.
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(36)	?	Is required to be set for a Credit CDR. This SHALL contain the `id` of the CDR for which this is a Credit CDR.
last_updated	DateTime	1	Timestamp when this CDR was last updated (or created).

NOTE The duration of charging (energy being transferred between EVSE and EV) during this session can be calculated via: `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, having a boolean flag that can be used for queries, is much faster then string comparisons.

9.3.1.1. Example of a CDR

```
{
  "id": "12345",
  "start_date_time": "2015-06-29T21:39:09Z",
  "stop_date_time": "2015-06-29T23:37:32Z",
  "token": {
    "uid": "012345678",
    "type": "RFID",
    "contract_id": "DE8ACC12E46L89",
    "visual_number": "DF000-2001-8999-1",
    "issuer": "TheNewMotion",
    "group_id": "DF000-2001-8999",
    "valid": true,
    "whitelist": "ALLOWED",
    "last_updated": "2015-06-29T22:39:09Z"
  },
  "auth_method": "WHITELIST",
  "cdr_location": {
    "id": "LOC1",
    "name": "Gent Zuid",
    "address": "F.Rooseveltlaan 3A",
    "city": "Gent",
    "postal_code": "9000",
    "country": "BE",
    "coordinates": {
      "latitude": "3.729944",
      "longitude": "51.047599"
    },
    "evses_uid": "3256",
    "evse_id": "BE*BEC*E041503003",
    "connectors_id": "1",
    "connectors_standard": "IEC_62196_T2",
    "connectors_format": "SOCKET",
    "connectors_power_type": "AC_1_PHASE"
  },
  "currency": "EUR",
  "tariffs": [{
    "id": "12",
    "currency": "EUR",
    "elements": [{
      "price_components": [{
        "type": "TIME",
        "price": 2.00,
        "vat": 10.0,
        "step_size": 300
      }]
    }],
    "last_updated": "2015-02-02T14:15:01Z"
  }],
  "charging_periods": [{
    "start_date_time": "2015-06-29T21:39:09Z",
    "dimensions": [{
      "type": "TIME",
      "volume": 1.973
    }],
    "tariff_id": "12"
  }],
  "total_cost": 4.00,
  "total_cost_incl_vat": 4.40,
  "total_energy": 15.342,
  "total_time": 1.973,
  "last_updated": "2015-06-29T22:01:13Z"
}
```

9.4. Data types

9.4.1. AuthMethod *enum*

Value	Description
AUTH_REQUEST	Authentication request from the eMSP

Value	Description
WHITELIST	Whitelist used to authenticate, no request done to the eMSP

9.4.2. CdrDimension *class*

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

9.4.3. CdrDimensionType *enum*

Value	Description
ENERGY	defined in kWh, default step_size is 1 Wh
FLAT	flat fee, no unit
MAX_CURRENT	defined in A (Ampere), Maximum current reached during charging session
MIN_CURRENT	defined in A (Ampere), Minimum current used during charging session
PARKING_TIME	time not charging: defined in hours, default step_size is 1 second
TIME	time charging: defined in hours, default step_size is 1 second

9.4.4. 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 CPOs 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)	1	Postal code of the location.
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 CPOs platform (and suboperator platforms). For example a database unique ID or the "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.
evse_id	CiString(48)	1	Compliant with the following specification for EVSE ID from "eMI3 standard version V1.0" (http://emi3group.com/documents-links/) "Part 2: business objects.".
connector_id	CiString(36)	1	Identifier of the connector within the EVSE.
connector_standard	ConnectorType	1	The standard of the installed connector.
connector_format	ConnectorFormat	1	The format (socket/cable) of the installed connector.
connector_power_type	PowerType	1	

9.4.5. CdrToken *class*

Property	Type	Card.	Description
uid	CiString(36)	1	Unique ID by which this Token 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 type=APP_USER Token, it will be a uniquely, by the MSP, 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 "eMI3 standard version V1.0" (http://emi3group.com/documents-links/) "Part 2: business objects."

9.4.6. 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. This period ends when a 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)	?	Uniquely identifier of the tariff that is relevant for this ChargingPeriod. If not provided, no tariff is relevant during this period.

10. *Tariffs* module

Module Identifier: `tariffs`

Data owner: `CPO`

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

10.1. Flow and Lifecycle

10.1.1. Push model

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

Any changes to the Tariff(s) in the CPO system can be send to the eMSP system by calling either PUT or PATCH 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 is deleted.

When the CPO is not sure about the state or existence of a Tariff object in the eMSPs system, the CPO can call the GET to validate the Tariff object in the eMSP system.

10.1.2. Pull model

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

10.2. Interfaces and endpoints

There is both a CPO and an eMSP interface for Tariffs. Advised is to use the push direction from CPO to eMSP during normal operation. The CPO interface is meant to be used when the connection between 2 parties is established to retrieve the current list of Tariffs objects, and when the eMSP is not 100% sure the Tariff cache is still correct.

10.2.1. CPO Interface

The CPO Tariffs interface gives the eMSP the ability to request tariffs.

Method	Description
GET	Returns Tariff Objects from the CPO, last updated 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 information about all Tariffs.

Endpoint structure definition:

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

Examples:

```
https://www.server.com/ocpi/cpo/2.2/tariffs/?date_from=2019-01-28T12:00:00&date_to=2019-01-
29T12:00:00
```

```
https://ocpi.server.com/2.2/tariffs/?offset=50
```

```
https://www.server.com/ocpi/2.2/tariffs/?date_from=2019-01-29T12:00:00&limit=100
```

```
https://www.server.com/ocpi/cpo/2.2/tariffs/?offset=50&amp;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 and date_to 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 this Date/Time.
date_to	DateTime	no	Only return Tariffs that have `last_updated` before this Date/Time.
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 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
Tariff	*	List of all tariffs.

10.2.2. eMSP Interface

Tariffs is a client owned object, so the end-points 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/tariffs/BE/BEC/12
```

Method	Description
GET	Retrieve a Tariff as it is stored in the eMSP system.
POST	n/a
PUT	Push new/updated Tariff object to the eMSP.
PATCH	Notify the eMSP of partial updates to a Tariff.

Method	Description
DELETE	Remove Tariff object which is no longer valid

10.2.2.1. GET Method

If the CPO wants to check the status of a Tariff 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.

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 this PUT to the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP 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.

10.2.2.2. 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.

Type	Card.	Description
Tariff	1	New or updated Tariff object

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 this PUT to the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
tariff_id	CiString(36)	yes	Tariff.id of the (new) Tariff object (to replace).

Example: New Tariff 2 euro per hour

```
PUT To URL: https://www.server.com/ocpi/emsp/2.2/tariffs/NL/TNM/12

{
  "id": "12",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "vat": 10.0,
      "step_size": 300
    }]
  }]
}
```

10.2.2.3. PATCH Method

The PATCH method works the same as the PUT method, except that the fields/objects that have to be updated have to be present, other fields/objects that are not specified are considered unchanged.

Example: Change Tariff to 2,50

```
PUT To URL: https://www.server.com/ocpi/emsp/2.2/tariffs/NL/TNM/12

{
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.50,
      "step_size": 300
    }]
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

10.2.2.4. DELETE Method

Delete a no longer valid Tariff object.

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 this PUT to the eMSP system.
party_id	CiString(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
tariff_id	CiString(36)	yes	Tariff.id of the Tariff object to delete.

10.3. Object description

10.3.1. *Tariff* Object

A Tariff Object consists of a list of one or more TariffElements, these elements can be used to create complex Tariff structures. When the list of *elements* contains more then 1 element, than the first tariff in the list with matching restrictions will be used.

It is advised to always set a "default" tariff, the last tariff in the list of *elements* with no restriction. This acts as a fallback when non of the TariffElements before this matches the current charging period.

To define a "Free of Charge" Tariff in OCPI, a tariff has to be provided that has a `type` = `FLAT` and `price` = `0.00`. See: Free of Charge Tariff example

Property	Type	Card.	Description
id	CiString(36)	1	Uniquely identifies the tariff within the CPOs platform (and suboperator platforms).
currency	string(3)	1	Currency of this tariff, ISO 4217 Code
type	TariffType	?	Defines what type of tariff this is. This makes it possible to make distinction possible for different Charging Preferences. When omitted, this tariff is valid for all sessions.
tariff_alt_text	DisplayText	*	List of multi language alternative tariff info text
tariff_alt_url	URL	?	Alternative URL to tariff info
min_price	Price	?	When this field is set, a Charging Session with this tariff will minimum cost this amount. This is different from a `FLAT` fee (Start Tariff, Transaction Fee), such a fee is a fixed amount that has to be payed for any Charging Session. A Minimum price means that when the cost of a Charging Session is lower then this amount, the cost of the Session will be equal to this amount. Also see note below.
max_price	Price	?	When this field is set, a Charging Session with this tariff will NOT cost more then this amount. See note below.
elements	TariffElement	+	List of tariff elements
start_datetime	DateTime	?	The time when this tariff becomes active. Typically used for a new tariff that is already given with the location, before it becomes active. (See note below)
end_datetime	DateTime	?	The time after which this tariff is no longer valid. 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 VAT might be built up of different parts, there might be situations where minimum including VAT is reached earlier or later then then the minimum excluding VAT. So as a rule: they both apply: - Minimum cost of a Charging Session including VAT can never be lower than the min_price including VAT. - Minimum cost of a Charging Session excluding VAT can never be lower than the min_price excluding VAT.

NOTE `max_price`: As the VAT might be built up of different parts, there might be situations where maximum including VAT is reached earlier or later then then the maximum excluding VAT. So as a rule: they both apply: - Total cost of a Charging Session including VAT can never be higher than the max_price including VAT. - Total cost of a Charging Session excluding VAT can never be higher than the max_price excluding VAT.

NOTE `start_datetime` and `end_datetime`: When the tariff of a Charge Point (Location) is changed with an ongoing charging session, it is common to not switch the tariff until the ongoing session is finished. But this is NOT a rule of OCPI. The driver has started charging and might have read the tariff info before charging. When the tariff is then changed during the session, the driver might get a bill that is higher then expected.

10.3.1.1. Examples

Simple Tariff example 0.25 euro per kWh

- 0.25 euro per kWh (excluding VAT)
- 10% VAT
- Billed per 1 Wh

```
{
  "id": "16",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "last_updated": "2018-12-17T11:16:55Z"
}
```

Tariff example 0.25 euro per kWh + start fee

- Start or transaction fee
    - 0.50 euro (excluding VAT)
    - 20% VAT
- Energy
    - 0.25 euro per kWh (excluding VAT)
    - 10% VAT
    - Billed per 1 Wh

This tariff will result in costs of 5.50 euro (ex VAT) when 20 kWh is charged.

```
{
  "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
    }]
  }],
  "last_updated": "2018-12-17T11:36:01Z"
}
```

Tariff example 0.25 euro per kWh + minimum price

- minimum price 0.50 (ex VAT)
- 0.25 euro per kWh (excluding VAT)
- 10% VAT
- Billed per 1 Wh

This tariff will result in costs of 5.00 euro (ex VAT) when 20 kWh is charged. But if less then 2 kWh is charged, 0.50 (ex VAT) will be billed.

This is different from a start fee, see example above.

```
{
  "id": "20",
  "currency": "EUR",
  "min_price": {
    "excl_vat": 0.50,
    "incl_vat": 0.55
  },
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.25,
      "vat": 10.0,
      "step_size": 1
    }]
  }],
  "last_updated": "2018-12-17T16:45:21Z"
}
```

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

- Start or transaction fee
  - 0.50 euro (excluding VAT)
  - 20% VAT
- Energy
  - 0.25 euro per kWh (excluding VAT)
  - 10% VAT
  - Billed per 1 Wh
- Parking
  - 2.00 euro per hour (excluding VAT)
  - 20% VAT
  - Billed per 15 min (900 seconds)

```
{
  "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
    }]
  }],
  "last_updated": "2018-12-17T11:44:10Z"
}
```

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

- Start or transaction fee

  - 0.50 euro (excluding VAT)
  - 20% VAT
- Energy
  - 0.25 euro per kWh (excluding VAT)
  - 10% VAT
  - Billed per 1 Wh

This tariff has a maximum price of 25 euro (ex VAT).

This tariff has an end date: 30 June 2019, this 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). By assigning both the old and the new tariff ID, they will automatically be replaced. It is then not need to update all location at the same time, the old tariff can be removed later.

```
{
  "id": "16",
  "currency": "EUR",
  "max_price": {
    "excl_vat": 25.00,
    "incl_vat": 27.50
  },
  "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
    }]
  }],
  "end_datetime": "2019-06-30T23:59:59Z",
  "last_updated": "2018-12-17T17:15:01Z"
}
```

Simple Tariff example 2 euro per hour

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

- 2.00 euro per hour charging (not per kWh) (excluding VAT)
- 10% VAT
- Billed per 1 minutes (60 seconds)

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

```
{
  "id": "12",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "vat": 10.0,
      "step_size": 60
    }]
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

Simple Tariff example 3 euro per hour, 5 euro 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 full.

- Charging time
  - 3.00 euro per hour charging (not per kWh) (excluding VAT)
  - 10% VAT
  - Billed per 1 minutes (60 seconds)
- Parking
  - 5.00 euro per hour (excluding VAT)
  - 20% VAT
  - Billed per 5 min (300 seconds)

```
{
  "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
    }]
  }],
  "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 less needed. Normally no CDR will be send to an MSP for ad-hoc charging. Having a good human readable text is much more useful.

- 2 euro per hour charging (not per kWh) (including VAT)
- 5.2% VAT
- Billed per 5 minutes (300 seconds)

```
{
  "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
    }]
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

Ad-Hoc Tariff example not possible with OCPI

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

Such a tariff cannot be modeled with OCPI, but by modeling it as 0.50 euro start tariff, using a debit card as a discount of 0.25 euro. Nobody is likely to complain. But the `tariff_alt_text` explains this clearly.

```
{
  "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
    }]
  }],
  "last_updated": "2018-17-29T15:55:58Z"
}
```

Simple Tariff example with alternative URL

This examples shows the use of `tariff_alt_url`.

This examples shows a `PROFILE_CHEAP` tariff. This is a smart-charging tariff. Were a driver selects to charge as cheaply as possible. 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, maybe you want to show a graph. In such a case, an URL can be given that links to a better explanation of the tariff.

- 0.25 euro per kWh (excluding VAT)

- 10% VAT
- Billed per 0.1 kWh (100 Wh)

```
{
  "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": 1
    }]
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

Complex Tariff example

- 2.50 euro start tariff
- 1.00 euro per hour charging tariff for less than 32A (paid per 15 minutes)
- 2.00 euro per hour charging tariff for more than 32A on weekdays (paid per 10 minutes)
- 1.25 euro per hour charging tariff for more than 32A during the weekend (paid per 10 minutes)

Parking costs:

- Weekdays: between 09:00 and 18:00 : 5 euro (paid per 5 minutes)
- Saturday: between 10:00 and 17:00 : 6 euro (paid per 5 minutes)

VAT:

- 15% on start tariff
- 20% on charging per hour
- 10% on parking

```
{
  "id": "14",
  "currency": "EUR",
  "type": "REGULAR",
  "tariff_alt_url": "https://company.com/tariffs/11",
  "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_power": 32.00
    }
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "vat": 20.0,
      "step_size": 600
    }],
    "restrictions": {
      "min_power": 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_power": 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"]
    }
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

Free of Charge Tariff example

In this example no VAT (that might not always be the case)

```
{
  "id": "15",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.00,
      "step_size": 0
    }]
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

First hour free energy example

- The first hour of parking time is free.
- From the second to the fourth hour is 2.00 euro per hour
- From the fourth on 3.00 euro per hour

Something similar applies to the kWh consumed.

- The first kWhs is free,
- 0.20 euro per kWh for the rest.

```
{
  "id" : "52",
  "currency" : "EUR",
  "elements" : [ {
    "price_components" : [ {
      "type" : "PARKING_TIME",
      "price" : 0.0,
      "step_size" : 60
    } ],
    "restrictions" : {
      "min_duration" : 0,
      "max_duration" : 3600
    }
  }, {
    "price_components" : [ {
      "type" : "PARKING_TIME",
      "price" : 2.0,
      "step_size" : 60
    } ],
    "restrictions" : {
      "min_duration" : 3600,
      "max_duration" : 10800
    }
  }, {
    "price_components" : [ {
      "type" : "PARKING_TIME",
      "price" : 3.0,
      "step_size" : 60
    } ],
    "restrictions" : {
      "min_duration" : 10800
    }
  }, {
    "price_components" : [ {
      "type" : "ENERGY",
      "price" : 0.0,
      "step_size" : 1
    } ],
    "restrictions" : {
      "max_kwh" : 1.0
    }
  }, {
    "price_components" : [ {
      "type" : "ENERGY",
      "price" : 0.2,
      "step_size" : 1
    } ],
    "restrictions" : {
      "min_kwh" : 1.0
    }
  } ]
}
```

Tariff example with reservation price

- Reservation
  - 5.00 euro per hour (excluding VAT)
  - 20% VAT
  - Billed per 1 minute (60 seconds)
- Start or transaction fee
  - 0.50 euro (excluding VAT)
  - 20% VAT
- Energy
  - 0.25 euro per kWh (excluding VAT)

    - 10% VAT
    - Billed per 1 Wh

```
{
  "id": "20",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 6.00,
      "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
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
}
```

Tariff example with reservation price and fee

- Reservation
    - 2.00 euro reservation fee (excluding VAT)
    - 5.00 euro per hour (excluding VAT)
    - 20% VAT
    - Billed per 5 minutes (300 seconds)
- Start or transaction fee
    - 0.50 euro (excluding VAT)
    - 20% VAT
- Energy
    - 0.25 euro per kWh (excluding VAT)
    - 10% VAT
    - Billed per 1 Wh

```
{
  "id": "20",
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 2.00,
      "vat": 20.0,
      "step_size": 1
    }, {
      "type": "TIME",
      "price": 6.00,
      "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
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
}
```

Tariff example with reservation price and expire fee

- Reservation
  - 4.00 euro reservation expire fee (excluding VAT) (when reservation not used, this fee is added to reservation cost)
  - 2.00 euro per hour (excluding VAT)
  - 20% VAT
  - Billed per 10 minutes (600 seconds)
- Start or transaction fee
  - 0.50 euro (excluding VAT)
  - 20% VAT
- Energy
  - 0.25 euro per kWh (excluding VAT)
  - 10% VAT
  - Billed per 1 Wh

```
{
  "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
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
}
```

Tariff example with reservation time and expire time

- Reservation
    - 3.00 euro per hour (excluding VAT)
    - 6.00 euro per hour (excluding VAT) (When reservation expires, EV driver never starts to charge)
    - 20% VAT
    - Billed per 10 minutes (600 seconds)
- Start or transaction fee
    - 0.50 euro (excluding VAT)
    - 20% VAT
- Energy
    - 0.25 euro per kWh (excluding VAT)
    - 10% VAT
    - Billed per 100 Wh

```
{
  "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
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
}
```

Example of cost:

- EV driver reserves the EVSE for half an hour (30 minutes).
- When the driver would start charging after 9 minutes, and charges 48.03 kWh: 0.50 + 0.50 + 12 = 13.00 euro (excluding VAT)
- When the driver would start charging after 12 minutes, and charges 62.98 kWh: 1.00 + 0.50 + 15.75 = 17.25 euro (excluding VAT)
- When the driver never start to charge before the reservation expires, cost: 3.00 euro (excluding VAT)

10.4. Data types

10.4.1. DayOfWeek *enum*

Value	Description
MONDAY	Monday
TUESDAY	Tuesday
WEDNESDAY	Wednesday
THURSDAY	Thursday
FRIDAY	Friday
SATURDAY	Saturday
SUNDAY	Sunday

10.4.2. PriceComponent *class*

Property	Type	Card.	Description
type	TariffDimensionType	1	Type of tariff dimension
price	number	1	price per unit (excluding VAT) for this tariff dimension
vat	number	?	applicable VAT percentage for this tariff dimension. If omitted, no VAT is applicable, that is different from 0% VAT, which would be a value of 0.0 here.
step_size	int	1	Minimum amount to be billed. This unit will be billed in this step_size blocks. For example: if type is time and step_size is 300, then time will be billed in blocks of 5 minutes, so if 6 minutes is used, 10 minutes (2 blocks of step_size) will be billed.

NOTE `step_size`: depends on the `type`, every `type` (except `FLAT`) defines a step_size multiplier, this is the size of every 'step' for that type in the gaven unit. For example: `PARKING_TIME` has 'step_size multiplier: 1 second' That means that the `step_size` of a `PriceComponent` is multiplied by 1 second. Thus a `step_size = 300` means 300 seconds = 5 minutes.

NOTE `step_size` shall only be taken into account for the last `TariffElement` and when switching to another `TariffElement` for the `PriceComponents` that are not in the new `TariffElement`. The same rule applies in case there is a switch between 2 tariffs (for example when a driver selects a different Charging Preference `profile_type`).

10.4.2.1. Examples tariff

Example tariff for explaining `step_size` when switching from one `TariffElement` to another:

- Charging cost 1.20 euro p/hour before 17:00, step_size: 30 minutes (1800 seconds)
- Charging cost 2.40 euro p/hour after 17:00, step_size: 15 minutes (900 seconds)
- Parking cost 1.00 euro p/hour before 20:00, step_size: 15 minutes (900 seconds)

```
{
  "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" : "24:00"
      }
    }
  ],
  "last_updated": "2018-12-18T17:07:11Z"
}
```

10.4.2.2. Example switch to different price:

An EV driver plugs in at 16:55, charges for 10 min (TIME), stops charging but stays plugged in for 2 min (PARKING TIME), and then leaves. Total session time = 12min.

- 5 billable minutes charging time before 17:00: 1.20 euro p/hour = 0.10 euro
- 10 billable minutes charging time after 17:00: 2.40 euro p/hour = 0.40 euro (10 minutes billed instead of 5: step_size is 15 minutes in the last tariff element, so total charging time is billed for 15 minutes)
- 15 billable minutes parking time: 1 euro p/hour = 0.25 euro
- 0.75 euro total for this session.

10.4.2.3. Example switching to free tariff element:

When parking becomes free after 20:00, the new `TariffElement` after 20:00 will not contain a `PARKING_TIME` `PriceComponent`. So the last parking period that needs to be payed, before 20:00, will be billed per step_size of the `PARKING_TIME` `PriceComponent` before 20:00.

Example of this:

An EV driver plugs in at 19:55, charges for 10 min (TIME), stops charging but stays plugged in for 2 min (PARKING TIME), and then leaves. Total session time = 12min.

- 5 billable minutes charging time before 20:00: 2.40 euro p/hour = 0.20 euro
- 10 billable minutes charging time after 20.00: 2.40 euro p/hour = 0.40 euro (10 minutes billed instead of 5: step_size is 15 minutes in the last tariff element, so total charging time is billed for 15 minutes)
- 15 billable minutes parking time: 1 euro p/hour = 0.25 euro (15 minutes billed instead of 5: step_size of the last PARKING_TIME is 900: 15 minutes)
- 0.85 euro total for this session.

10.4.3. ReservationRestrictionType *enum*

Value	Description
RESERVATION	This TariffElement is for the cost associated with a reservation.
RESERVATION_EXPIRES	This TariffElement is only for cost associated with a reservation that expires, driver does not start a charging session before expiry_date.

NOTE When a Tariff has both a `RESERVATION` and a `RESERVATION_EXPIRES` TariffElement, with both a TIME PriceComponent, the time based cost of an expired reservation duration will be the price in the `RESERVATION_EXPIRES` TariffElement.

10.4.4. TariffElement *class*

Property	Type	Card.	Description
price_components	PriceComponent	+	List of price components that make up the pricing of this tariff
restrictions	TariffRestrictions	?	Tariff restrictions object

10.4.5. TariffDimensionType *enum*

Value	Description
ENERGY	defined in kWh, step_size multiplier: 1 Wh
FLAT	flat fee, no unit
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 reservation for the the price of the reservation time.

10.4.6. TariffRestrictions *class*

Property	Type	Card.	Description
start_time	string(5)	?	Start time of day, for example 13:30, valid from this time of the day. Must be in 24h format with leading zeros. Hour/Minute separator: ":" Regex: [0-2][0-9]:[0-5][0-9]
end_time	string(5)	?	End time of day, for example 19:45, valid until this time of the day. Same syntax as start_time
start_date	string(10)	?	Start date, for example: 2015-12-24, valid from this day
end_date	string(10)	?	End date, for example: 2015-12-27, valid until this day (excluding this day)
min_kwh	number	?	Minimum used energy in kWh, for example 20, valid from this amount of energy is used
max_kwh	number	?	Maximum used energy in kWh, for example 50, valid until this amount of energy is used
min_power	number	?	Minimum power in kW, for example 0, valid from this charging speed
max_power	number	?	Maximum power in kW, for example 20, valid up to this charging speed
min_duration	int	?	Minimum duration in seconds, valid for a duration from x seconds
max_duration	int	?	Maximum duration in seconds, valid for a duration up to x seconds
day_of_week	DayOfWeek	*	Which day(s) of the week this tariff is valid
reservation	ReservationRestrictionType	?	When this field is present, this tariffElement is for a reservation. A reservation starts when the reservation is made, and ends when the drivers start 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.

10.4.7. TariffType *enum*

Value	Description
AD_HOC_PAYMENT	This tariff is valid when ad hoc payment is used at the Charge Point. Instead of an RFID token or APP.
PROFILE_CHEAP	This tariff is valid when Charging Preference: CHEAP is set on the session.
PROFILE_FAST	This tariff is valid when Charging Preference: FAST is set on the session.
PROFILE_GREEN	This tariff is valid when Charging Preference: GREEN is set on the session.
REGULAR	This is the tariff when using an RFID, without any Charging Preference, or when Charging Preference: REGULAR is set on the session.

11. *Tokens* module

Module Identifier: `tokens`

Data owner: `MSP`

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.

11.1. Flow and Lifecycle

11.1.1. Push model

When the MSP 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.

11.1.2. 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) every X hours.

11.1.3. 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`.

11.2. Interfaces and endpoints

There is both a CPO and an eMSP interface for Tokens. It is advised to use the push direction from eMSP to CPO during normal operation. The eMSP interface is meant to be used when the CPO is not 100% sure the Token cache is still correct.

11.2.1. CPO Interface

With this interface the eMSP can push the Token information to the CPO. 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}
```

Example:

```
https://www.server.com/ocpi/cpo/2.2/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).

11.2.1.1. 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 can be provided as URL segments.

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.

Response Data

The response contains the requested object.

Type	Card.	Description
Token	1	The requested Token object.

11.2.1.2. 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 can 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.
token_uid	CiString(36)	yes	Token.uid of the (new) Token object (to replace).

Example: put a new Token

```
PUT To URL: https://www.server.com/ocpi/cpo/2.2/tokens/NL/TNM/012345678

{
  "uid": "012345678",
  "type": "RFID",
  "contract_id": "DE8ACC12E46L89",
  "visual_number": "DF000-2001-8999-1",
  "issuer": "TheNewMotion",
  "group_id": "DF000-2001-8999",
  "valid": true,
  "whitelist": "ALWAYS",
  "last_updated": "2015-06-29T22:39:09Z"
}
```

11.2.1.3. 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.

Example: invalidate a Token

```
PATCH To URL: https://www.server.com/ocpi/cpo/2.2/tokens/NL/TNM/012345678

{
  "valid": false
}
```

11.2.2. eMSP Interface

This interface enables the CPO 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

11.2.2.1. 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={limi
t}]
```

Examples:

```
https://www.server.com/ocpi/emps/2.2/tokens/?date_from=2019-01-28T12:00:00&date_to=2019-01-
29T12:00:00
```

```
https://ocpi.server.com/2.2/tokens/?offset=50
```

```
https://www.server.com/ocpi/2.2/tokens/?date_from=2019-01-29T12:00:00&limit=100
```

```
https://www.server.com/ocpi/emsp/2.2/tokens/?offset=50&amp;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 and date_to 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 this Date/Time.
date_to	DateTime	no	Only return Tokens that have `last_updated` before this Date/Time.
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.

11.2.2.2. 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=token_type}]
```

The `/authorize` is required for the real-time authorize request.

Examples:

```
https://www.server.com/ocpi/emps/2.2/tokens/012345678/authorize
```

```
https://ocpi.server.com/2.2/tokens/012345678/authorize?type=RFID
```

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 SHOULD respond with the OCPI status: 2002

Request Parameters

The following parameter has to be provided as URL segments.

Parameter	Datatype	Required	Description
token_uid	CiString(36)	yes	Token.uid of the Token for which this authorization is.
token_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/Connectors. The object with valid Location and EVSEs/Connectors will be returned in the response.

Type	Card.	Description
LocationReferences	?	Location and EVSEs/Connectos for which the Token is requested to be authorized.

Response Data

The endpoint response contains a AuthorizationInfo object.

Type	Card.	Description
AuthorizationInfo	1	Contains information about the authorization, if the Token is allowed to charge and optionally which EVSEs/Connectors are allowed to be used.

11.3. Object description

11.3.1. *AuthorizationInfo* Object

Property	Type	Card.	Description
allowed	Allowed	1	Status of the Token, and whether charging is allowed at the optionally given location.
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/Connectors the EV driver is allowed to charge at are returned.
info	DisplayText	?	Optional display text, additional information to the EV driver.

11.3.2. *Token* Object

Property	Type	Card.	Description
uid	CiString(36)	1	Unique ID by which this Token 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 type=APP_USER Token, it will be a uniquely, by the MSP, generated ID. This field is named `uid` instead of `id` to prevent confusion with: `contract_id`.
type	TokenType	1	Type of the token

Property	Type	Card.	Description
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 "eMI3 standard version V1.0" (http://emi3group.com/documents-links/) "Part 2: business objects."
visual_number	string(64)	?	Visual readable number/identification as printed on the Token (RFID card), might be equal to the contract_id.
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.

11.3.2.1. Examples

Simple APP_USER example

```
{
  "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

```
{
  "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"
}
```

11.4. Data types

11.4.1. Allowed *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.

11.4.2. 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.

11.4.3. 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 identifier for EVSEs within the CPO's platform for the EVSE within the the given location.
connector_ids	CiString(36)	*	Identifies the connectors within the given EVSEs.

11.4.4. TokenType *enum*

Value	Description
APP_USER	Token ID generated by a server (or App.) to identify a user of an App.

Value	Description
OTHER	Other type of token
RFID	RFID Token

11.4.5. WhitelistType *enum*

Defines when authorization of a Token by the CPO is allowed.

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.

12. *Commands* module

Module Identifier: `commands`

The Commands module enables remote commands to be sent to a Location/EVSE. The following commands are supported:

- `RESERVE_NOW`
- `CANCEL_RESERVATION`
- `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

12.1. Flow

With the Commands module, commands can be sent from the eMSP, via the CPO to a Charge Point. Most Charge Point 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 eMSP send a request to a CPO, via the CPO Commands interface. The CPO 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 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 CPO will forward the result in a new POST request to the eMSP 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.

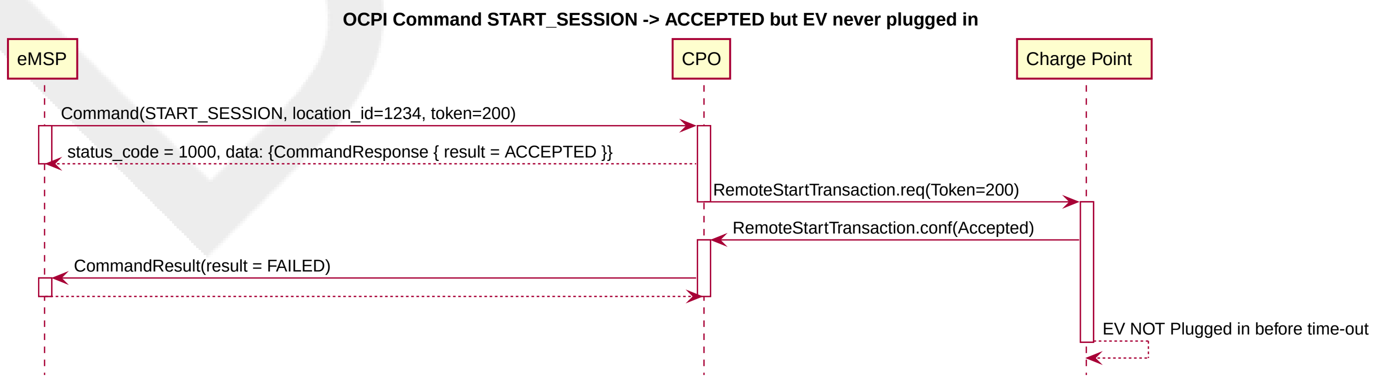
OCPI Command START_SESSION -> ACCEPTED but EV never plugged in
eMSP
CPO
Charge Point
Command(START_SESSION, location_id=1234, token=200)
status_code = 1000, data: {CommandResponse { result = ACCEPTED }}
RemoteStartTransaction.req(Token=200)
RemoteStartTransaction.conf(Accepted)
CommandResult(result = FAILED)
EV NOT Plugged in before time-out

Figure 9. 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.

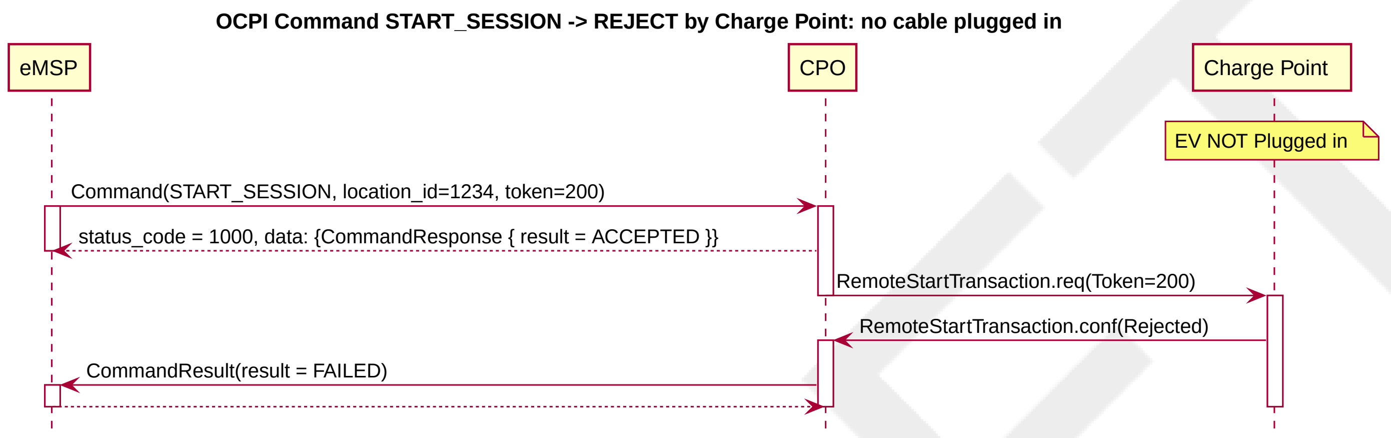
OCPI Command START_SESSION -> REJECT by Charge Point: no cable plugged in
eMSP
CPO
Charge Point
EV NOT Plugged in
Command(START_SESSION, location_id=1234, token=200)
status_code = 1000, data: {CommandResponse { result = ACCEPTED }}
RemoteStartTransaction.req(Token=200)
RemoteStartTransaction.conf(Rejected)
CommandResult(result = FAILED)

Figure 10. START_SESSION failed

Example of a `START_SESSION` that is accepted and results in a new Session.

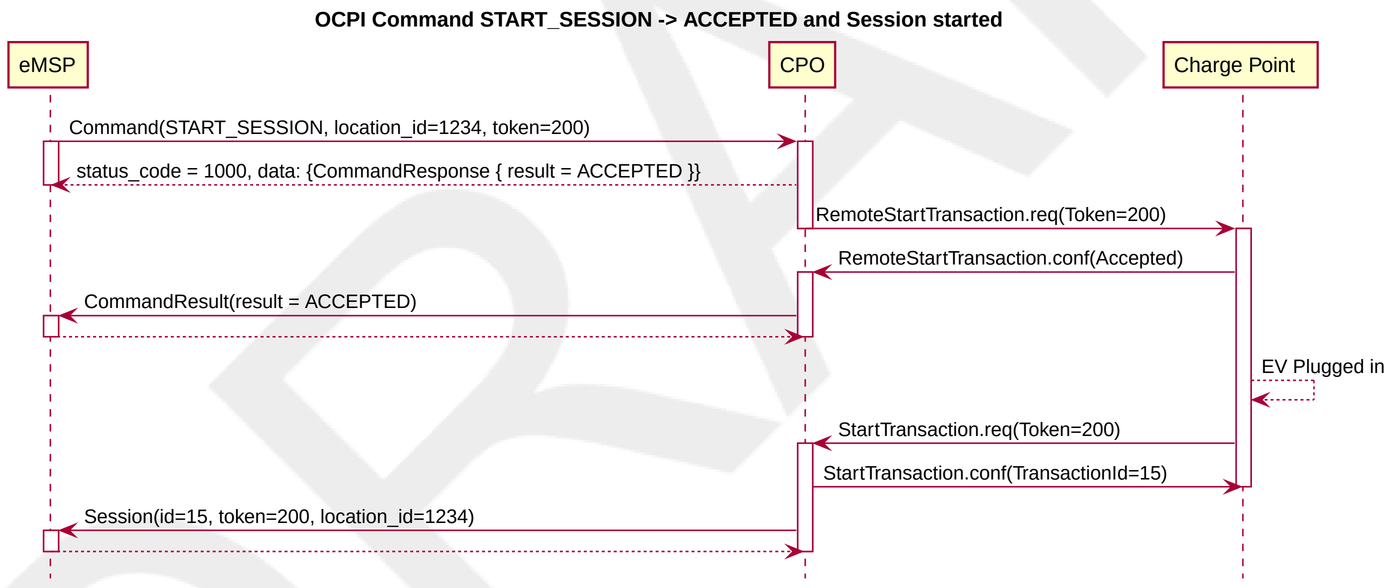
OCPI Command START_SESSION -> ACCEPTED and Session started
eMSP
CPO
Charge Point
Command(START_SESSION, location_id=1234, token=200)
status_code = 1000, data: {CommandResponse { result = ACCEPTED }}
RemoteStartTransaction.req(Token=200)
RemoteStartTransaction.conf(Accepted)
CommandResult(result = ACCEPTED)
EV Plugged in
StartTransaction.req(Token=200)
StartTransaction.conf(TransactionId=15)
Session(id=15, token=200, location_id=1234)

Figure 11. START_SESSION successful

Example of a `UNLOCK_CONNECTOR` that fails because the Location is not known by the CPO.

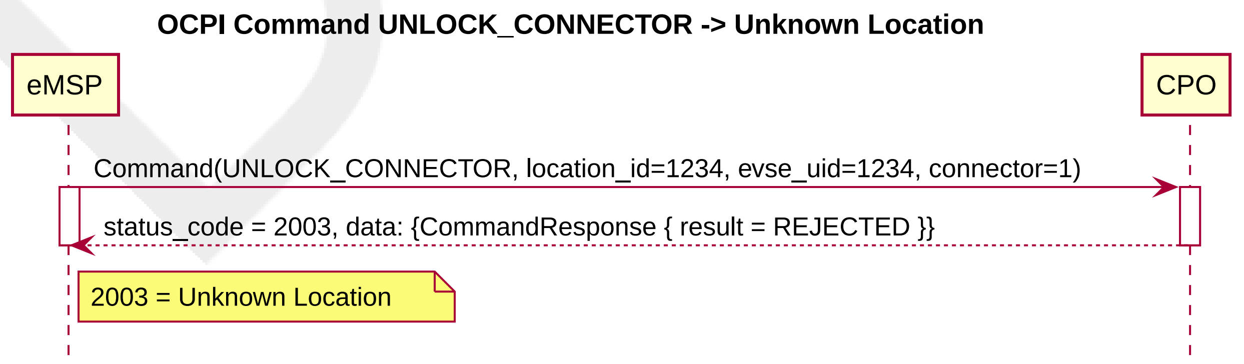
OCPI Command UNLOCK_CONNECTOR -> Unknown Location
eMSP
CPO
Command(UNLOCK_CONNECTOR, location_id=1234, evse_uid=1234, connector=1)
status_code = 2003, data: {CommandResponse { result = REJECTED }}
2003 = Unknown Location

Figure 12. UNLOCK_CONNECTOR Unknown Location

Example of a `RESERVE_NOW` that is rejected by the Charge Point.

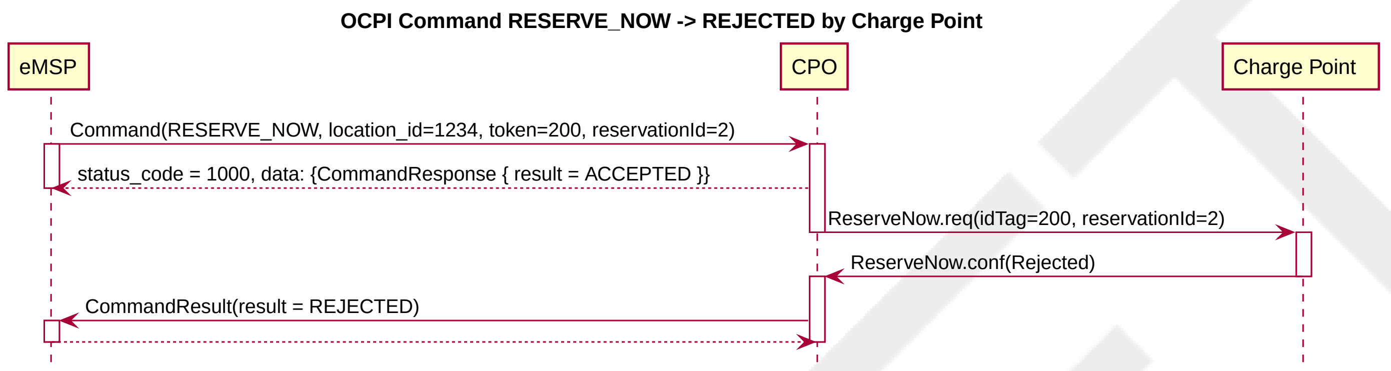
OCPI Command RESERVE_NOW -> REJECTED by Charge Point
eMSP
CPO
Charge Point
Command(RESERVE_NOW, location_id=1234, token=200, reservationId=2)
status_code = 1000, data: {CommandResponse { result = ACCEPTED }}
ReserveNow.req(idTag=200, reservationId=2)
ReserveNow.conf(Rejected)
CommandResult(result = REJECTED)

Figure 13. RESERVE_NEW rejected by Charge Point

These examples use OCPP 1.6 based commands between CPO and Charge Point, but that is not a requirement for OCPI.

12.2. Interfaces and endpoints

The commands module consists of two interfaces: a CPO interface that enables a eMSP (and its clients) to send commands to a Location/EVSE and an eMSP interface to receive the response from the Location/EVSE asynchronously.

12.2.1. CPO Interface

Endpoint structure definition:

`{commands_endpoint_url}{command}`

Examples:

`https://www.server.com/ocpi/cpo/2.2/commands/START_SESSION`

`https://ocpi.server.com/commands/STOP_SESSION`

`https://server.com/ocpi/cpo/2.2/commands/RESERVE_NOW`

Example endpoint structure: ``

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

12.2.1.1. POST Method

Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
command	CommandType	yes	Type of command that is requested.

12.2.1.2. 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 CPO, not the response from the Charge Point itself, that will be sent via an asynchronous POST on the eMSP 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

12.2.2. eMSP Interface

The eMSP interface receives the asynchronous responses.

Endpoint structure definition:

No structure defined. This is open to the MSP to define, the URL is provided to the CPO by the MSP in the POST to the CPO interface. Therefor OCPI does not define variables.

Example:

```
https://www.server.com/ocpi/emsp/2.2/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
PATCH	n/a
DELETE	n/a

12.2.2.1. 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/commands/RESERVE_NOW/1234
```

```
https://www.server.com/ocpi/emsp/2.2/commands/UNLOCK_CONNECTOR/2
```

12.2.2.2. Request Body

Datatype	Card.	Description
CommandResult	1	Result of the command request, from the Charge Point.

12.3. Object description

12.3.1. *CancelReservation* Object

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResponse POST should be send to. This URL might contain an unique ID to be able to distinguish between ReserveNow requests.
reservation_id	int	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.

12.3.2. *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 known 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.

12.3.3. *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.

12.3.4. *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`.

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResponse POST should be send to. This URL might contain an 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.
reservation_id	int	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.
location_id	CiString(36)	1	Location.id of the Location (belonging to the CPO this request is send 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.

12.3.5. *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).

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResponse POST should be sent to. This URL might contain an 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.
location_id	CiString(36)	1	Location.id of the Location (belonging to the CPO this request is send 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.

12.3.6. *StopSession* Object

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResponse POST should be sent to. This URL might contain an unique ID to be able to distinguish between StopSession requests.
session_id	CiString(36)	1	Session.id of the Session that is requested to be stopped.

12.3.7. *UnlockConnector* Object

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResponse POST should be sent to. This URL might contain an 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 send 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.

12.4. Data types

12.4.1. CommandResponseType *enum*

Response to the command request from the MSP 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 MSP 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.

12.4.2. CommandResultType *enum*

Result of the command that was send to the Charge Point.

Value	Description
ACCEPTED	Command request accepted by the Charge Point.
EVSE_OCCUPIED	EVSE is currently occupied, another session is ongoing. Cannot start a new session
EVSE_INOPERATIVE	EVSE is currently inoperative or faulted.
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.

12.4.3. CommandType *enum*

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.

13. *ChargingProfiles* module

Module Identifier: `charging_profiles`

With the ChargingProfiles module, parties (MSPs) can send (Smart) Charging Profiles to a Location/EVSE. It is also possible to request the 'CompositeProfile' from a Location/EVSE.

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 garantue that the EV will charge with the exact given limit. 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 to warm. The cable might be one phase on a three phase charger etc. There can be local energy limits (load balancing between EVSE on a relative small energy connection to a group of EVSEs) that limit the energy offered by 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 MSP, 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 oblige local privacy laws, have to have setup contracts etc. Local laws might oblige explicit consent from the driver etc.

The CompositeProfile is the charging schedule 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.

Module dependency: Sessions module

13.1. Use Cases

TODO

- An eMSP sends a ChargingProfile to manipulate an ongoing charging session of one of its customers.
- An eMSP request the planned ChargingProfile for an ongoing charging session for a customer of the MSP.
- An eMSP request to remove the set ChargingProfile from an ongoing charging session for a customer of the MSP.
- The CPO updates the eMSP of changes to an CompositeSchedule.
- Energy contract meenemen naar publieke laadpaal -

13.2. 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 eMSP send a request to a CPO, via the CPO Commands interface. The CPO 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 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 ChargingProfile will be executed. The CPO will forward the result in a new POST request to the eMSP ChargingProfile interface.

The eMSP can send the Charging Profile to the EVSE via the CPO by using the CPO PUT method for an ongoing session. The eMSP can request the current schedule 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 eMSP has the ability to remove the Charging Profile for the session by calling the CPO DELETE method

When the eMSP has (at least once) successfully send a Charging Profile for an ongoing charging session, the CPO keeps the eMSP updated of changes to the CompositeSchedule, if the CPO is aware of any changes, by calling the MSP PUT method

The CPO can cancel an existing ChargingProfile, it can let the eMSP know by calling the MSP PUT method

TODO Added examples for:

- GetCompositeProfile
- SetChargingProfile
- ClearChargingProfile
- UpdateCompositeSchedule

TODO Add examples

13.3. Interfaces and endpoints

The ChargingProfiles module consists of two interfaces: a CPO interface that enables a eMSP (and its clients) to send ChargingProfiles to a Location/EVSE, and an eMSP interface to receive the response from the Location/EVSE asynchronously.

13.3.1. CPO Interface

ChargingProfiles is a client owned object, so the end-points need to contain the required extra fields: {party_id} and {country_code}.

Example endpoint structures:

Method	Description
GET	Gets the active ChargingProfile for a specific charging session.
POST	n/a
PUT	Creates/updates a ChargingProfile for a specific charging session.
PATCH	n/a
DELETE	Cancels an existing ChargingProfile for a specific charging session.

13.3.1.1. GET Method

Retrieves the Composite ChargingProfile 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/chargingprofiles/1234?duration=900&response_url=https://www.msp.com
/ocpi/2.2/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 can be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the CPO platform.
duration	int	1	Length of the requested CompositeSchedule in seconds Duration in seconds. *
response_url	URL	1	URL that the CompositeProfileResult POST should be send to. This URL might contain an unique ID to be able to distinguish between GET Composite ChargingProfile requests.

NOTE duration: Use this wisely. Asking for a schedule hours in advance might not be very useful. But will use more mobile data then really useful. Duration of half hour, one full hour can be really useful when checking the profile calculated by the Charging Station. Longer might be less useful as lot can change during the time that will have influence on the profile.

Response Data

The response contains the direct response from the CPO, not the response from the EVSE itself, that will be sent via an asynchronous POST on the eMSP interface if this response is `ACCEPTED`.

Datatype	Card.	Description
ChargingProfileResponse	1	Result of the Composite ChargingProfile request, by the CPO (not the location/EVSE). So this indicates if the CPO understood the ChargingProfile request and was able to send it to the EVSE. This is not the response by the Charge Point.

13.3.1.2. 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/chargingprofiles/1234`

Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the CPO platform.

13.3.1.3. Request Body

The body contains an 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 CPO, not the response from the EVSE itself, that will be sent via an asynchronous POST on the eMSP 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.

13.3.1.4. DELETE Method

Clears the ChargingProfile set by the MSP on the given session.

Endpoint structure definition:

`{chargingprofiles_endpoint_url}{session_id}?response_url={url}`

Example:

`https://www.cpo.com/ocpi/2.2/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 can be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the CPO platform.
response_url	URL	1	URL that the ClearProfileResult POST should be send to. This URL might contain an unique ID to be able to distinguish between GET Composite ChargingProfile requests.

Response Data

The response contains the direct response from the CPO, not the response from the EVSE itself, that will be sent via an asynchronous POST on the eMSP 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.

13.3.2. eMSP Interface

The eMSP interface receives the asynchronous responses.

Method	Description
GET	n/a
POST	Receive the asynchronous response from the Charge Point.
PUT	CPO can send an updated composite schedule when other inputs have made changes to existing schedule. When the CPO sends a update schedule to the EVSE, for an other reason then the MSP, the CPO SHALL post an update to this interface. When a local input influence the CompositeSchedule in the EVSE AND the CPO is made aware of this, the CPO SHALL post an update to this interface.
PUT	n/a
PATCH	n/a
DELETE	n/a

13.3.2.1. POST Method

Request Parameters

There are no URL segment parameters required by OCPI.

As the eMSP interface is called by the CPO on the URL given `response_url` in the MSP request to the CPO interface, 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.

Endpoint structure definition:

No structure defined. This is open to the MSP to define, the URL is provided to the CPO by the MSP. Therefor OCPI does not define variables.

Examples:

```
https://www.server.com/ocpi/2.2/chargingprofiles/chargingprofile/12345678
```

```
https://www.server.com/compositeschedule/12345678
```

```
https://www.server.com/clearprofile?request_id=12345678
```

```
https://www.server.com/ocpi/2.2/12345678
```

The content of the request body depends on the original request by the MSP to which this POST is send as a result.

13.3.2.2. Request Body

Datatype	Card.	Description
Choice: one of three		
CompositeProfileResult	1	Result of the GET CompositeProfile request, from the Charge Point.
ChargingProfileResult	1	Result of the PUT ChargingProfile request, from the Charge Point.
ClearProfileResult	1	Result of the DELETE ChargingProfile request, from the Charge Point.

13.3.2.3. Response Body

The response to the POST on the eMSP interface SHALL contain the Response Format with the data field omitted.

13.3.2.4. PUT Method

Updates the eMSP when the CPO knows the CompositeSchedule has changed.

The CPO SHALL call this interface every time it knows changes have been made that influence the CompositeProfile for an ongoing session AND the eMSP has at least once successfully called the charging profile CPO PUT interface for this session (SetChargingProfile). If the CPO doesn’t know the composite schedule has changed (EVSE does not notify the CPO of the change) it is not required to call this interface.

The CPO SHALL NOT call this interface for any session where the eMSP has never, successfully called the charging profile CPO PUT interface for this session (SetChargingProfile).

The CPO SHALL send a useful relevant duration of CompositeSchedule to send to the eMSP. As a guide: between 5 and 60 minutes. If the eMSP wants a longer CompositeSchedule the eMSP 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/chargingprofiles/1234`
```

Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
session_id	CiString(36)	yes	The unique id that identifies the session in the CPO platform.

13.3.2.5. Request Body

The body contains an SetChargingProfile object, that contains the new ChargingProfile and a response URL.

Type	Card.	Description
schedule	CompositeProfile	1

13.3.2.6. Response Body

The response to the PUT on the eMSP interface SHALL contain the Response Format with the data field omitted.

13.4. Object description

13.4.1. *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 known the timeout on a certain command.

Property	Type	Card.	Description
result	ResponseType	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 send.

13.4.2. *CompositeProfileResult* Object

The CompositeProfileResult object is send by the CPO to the given `response_url` in a POST request. It contains the result of the GET (GetCompositeProfile) request send by the MSP.

Property	Type	Card.	Description
result	ResultType	1	The EVSE will indicate if it was able to process the request for the Composite Profile
schedule	CompositeProfile	?	The requested composite profile, if the result field is set to: `ACCEPTED`

13.4.3. *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 MSP.

Property	Type	Card.	Description
result	ResultType	1	The EVSE will indicate if it was able to process the new/updated charging profile.

13.4.4. *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 MSP.

result	ResultType	1	The EVSE will indicate if it was able to process the removal of the charging profile (ClearChargingProfile).

13.4.5. *SetChargingProfile* Object

Object set to a CPO to set a Charging Profile.

Property	Type	Card.	Description
charging_schedule	[mod_charging_profiles_charging_schedule_object]	1	Contains limits for the available power or current over time.
response_url	URL	1	URL that the ChargingProfileResult POST should be send to. This URL might contain an unique ID to be able to distinguish between GET Composite ChargingProfile requests.

13.5. Data types

13.6. ChargingRateUnit *enum*

Unit in which a charging schedule is defined.

Value	Description
W	Watts (power) This is the TOTAL allowed charging power. If used for AC Charging, the phase current should be calculated via: Current per phase = Power / (Line Voltage * Number of Phases). The "Line Voltage" used in the calculation is not the measured voltage, but the set voltage for the area (hence, 230 of 110 volt). The "Number of Phases" is the numberPhases from the ChargingSchedulePeriod. It is usually more convenient to use this for DC charging. Note that if numberPhases in a ChargingSchedulePeriod is absent, 3 SHALL be assumed.
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.

13.7. ChargingSchedule *class*

Charging schedule class defines a list of charging periods.

Property	Type	Card.	Description
start_schedule_period	DateTime	?	Starting point of an absolute schedule. If absent the schedule will be relative to start of charging.
duration	int	?	Duration of the charging schedule in seconds. If the duration is left empty, the last period will continue indefinitely or until end of the transaction in case startSchedule is absent.
charging_rate_unit	ChargingRateUnit	1	The unit of measure Limit is expressed in.
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_schedule_period	ChargingSchedulePeriod	*	List of ChargingSchedulePeriod elements defining maximum power or current usage over time.

13.8. ChargingSchedulePeriod *class*

Charging schedule period structure defines a time period in a charging schedule, as used in: ChargingSchedule

Property	Type	Card.	Description
start_period	int	1	Start of the period, in seconds from the start of schedule. The value of StartPeriod also defines the stop time of the previous period.
limit*	number	1	Charging rate limit during the schedule period, in the applicable chargingRateUnit, for example in Amperes (A) or Watts (W). Accepts at most one digit fraction (e.g. 8.1).

13.8.1. CompositeProfile *class*

Property	Type	Card.	Description
start_date_time	DateTime	1	Date and time at which the schedule becomes active. All time measurements within the schedule are relative to this timestamp.
charging_schedule	ChargingSchedule	1	Charging schedule structure defines a list of charging periods.

13.8.2. ResponseType *enum*

Response to the ChargingProfile request from the MSP to the CPO.

Value	Description
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 MSP that send this request)
ACCEPTED	ChargingProfile request accepted by the CPO, request will be forwarded to the EVSE.
UNKNOWN_SESSION	The Session in the requested command is not known by this CPO.

13.8.3. ResultType *enum*

Result of a ChargingProfile request that the EVSE sends via the CPO to the MSP.

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.

14. *HubClientInfo* module

Module Identifier: `hubclientinfo`

Data owner: `Hub`

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 eMPS and other parties are online and vice versa.

It is like any other OCPI module, but then not between eMSP and CPO, but between eMSP/CPO and Hub.

14.1. Scenarios

This section will describe what the expected behavior is when a party receive information of a ConnectionState change.

14.1.1. Another Party becomes CONNECTED

Party is (back) online. Request can be send again. Every party receiving Client Owned Object from this party should be prepared to received client owned objects with URLs that contain the party_id and country_code of this party.

14.1.2. Another Party goes OFFLINE

Connection to party is not available: No requests can be send. Do not queue push messages. When the other parties comes back online, it is their responsibility to do a GET to get back in sync.

14.1.3. Another Party becomes PLANNED

No requests can be send to this new party yet. Might be a good idea to send 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 send to parties.

14.1.4. Another Party becomes SUSPENDED

Like with OFFLINE, no requests should be send to this party, they cannot be delivered.

When for example CDRs still have to be delivered, so there is some unfinished business, parties are advised to get into contact with the other party in another way the OCPI: Call them, send an e-mail.

14.2. 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 5 minutes (TODO determine the best time for a keep alive) 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.

14.3. Flow and Life-cycle

14.4. 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 send to the connected party system by calling, either the PUT or the PATCH 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, either the PUT or the PATCH 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.

14.5. 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.

14.6. Interfaces

There is both a Hub interface as a connected client (eMSP/CPO etc) interface for ClientInfo. It is advised to use the push direction from Hub 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.

14.6.1. Connected client Interface

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	Notify the connected client of partial updates to a ClientInfo object.
DELETE	n/a, Use PUT, ClientInfo objects cannot be removed).

14.6.1.1. 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 can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	CiString(2)	yes	Country code of the requested ClientInfo object.

Parameter	Datatype	Required	Description
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.

14.6.1.2. 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 can 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",
}
```

14.6.1.3. 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.

Example: invalidate a ClientInfo object

```
PATCH To URL: https://www.server.com/ocpi/cpo/2.0/clientinfo/NL/ALL

{
  "status": "SUSPENDED"
}
```

14.6.2. Hub Interface

This interface enables a connected party to request the current list of ClientInfo objects, 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

14.6.2.1. 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/hubclientinfo/?date_from=2019-01-28T12:00:00&date_to=2019-01-29T12:00:00`

`https://ocpi.server.com/2.2/hubclientinfo/?offset=50`

`https://www.server.com/ocpi/2.2/hubclientinfo/?date_from=2019-01-29T12:00:00&limit=100`

`https://www.server.com/ocpi/cpo/2.2/hubclientinfo/?offset=50&amp;limit=100`

14.6.2.2. Request Parameters

If additional parameters: {date_from} and/or {date_to} are provided, only ClientInfo objects with (`last_updated`) between the given date_from and date_to 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 this Date/Time.
date_to	DateTime	no	Only return ClientInfo that have `last_updated` before this Date/Time.
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

14.6.2.3. 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.

14.6.3. Object description

14.7. *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	RoleType	1	The role of the connected party.
status	ConnectionStatusType	1	Status of the connection to the party.
last_updated	DateTime	1	Timestamp when this ClientInfo object was last updated.

15. Data types

15.1. RoleType *enum*

Value	Description
CPO	Charging Point Operator.
EMPS	e-Mobility Service Provider.
NSP	Navigation Service Provider, role like an eMSP (probably only interested in Location information)
OTHER	Other role

15.2. ConnectionStatusType *enum*

Value	Description
CONNECTED	Party is connected.
OFFLINE	Party is currently not connected.
PLANNED	Connection to this party is planned, but has never been connected.
SUSPENDED	Party is now longer active, will never connect anymore.

16. Types

16.1. CiString *type*

Case Insensitive String. Only printable ASCII allowed.

16.2. DateTime *type*

All timestamps are formatted as string(25) using the combined date and time format from the ISO 8601 standard. All timestamps SHALL be in UTC. The absence of the timezone designator implies a UTC timestamp.

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
2016-12-29T17:45:09Z
2016-12-29T17:45:09
2018-01-01T01:08:01Z
2018-01-01T01:08:01
```

Note: +00:00 is not the same as UTC.

16.3. 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"
}
```

16.4. number *type*

Numbers in OCPI are formatted as JSON numbers. Unless mentioned otherwise, numbers use 4 decimals and a *sufficiently large amount* of digits.

16.5. Price *class*

Property	Type	Card.	Description
excl_vat	number	1	Price/Cost excluding VAT.
incl_vat	number	1	Price/Cost including VAT.

16.6. string *type*

Case Sensitive String. Only printable ASCII allowed. All strings in messages and enumerations are case sensitive, unless explicitly stated otherwise.

16.7. URL *type*

An URL a string(255) type following the w3.org spec.

17. Changelog

17.1. Changes between OCPI 2.1.1 and 2.2

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
CDRs / CDR Object	Minor / Minor	Minimal / Minimal	- Added `session_id` field, making it easier to match a CDR to a Session. - Changed `total_cost` field from type: `number` to `Price`, this provides the MSP 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 deticated 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. - 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.
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 / Minor	Minimal / Minimal	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.
Commands / StartSession Object	Minor / Minor	Minimal / Minimal	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.
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.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
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 / CPO 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 / CPO GET, PUT & PATCH 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.
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.
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.
Locations / Location Object	Minor / Minor	Minimal / Minimal	- 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.
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 / 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 / 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.
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.
Sessions / CPO 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,

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Sessions / eMSP GET, PUT, PATCH 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 / Session Object	Minor / Minor	Minimal / Minimal	- 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 MSP with the total cost including VAT. - 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.
Tokens / eMSP GET, PUT, PATCH & DELETE 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.
Tariffs / Tariff 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 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.
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).
Tokens / CPO GET & PUT 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 / eMSP POST method	Minor / Minor	Minimal / Minimal	Changed `token_uid` from string to CiString, making it case-insensitive, which had always been the idea.
Tokens / Token Object	Minor / Minor	Minimal / Minimal	- 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 / LocationReferences class	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.
Tokens / TokenType enum	Minor / Minor	Minimal / Minimal	Added value `APP_USER`. As more and more MSPs are launching Apps, this becomes more common, so a special category is useful.
Versions / Endpoint class	Medium / Medium	Minimal / Minimal	Field `role` added, making it possible to have one OCPI version end-point for both MSP and CPO role, so one OCPI connection when both CPO and MSP 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.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Transport & Format	Minor / Minor	Minimal / Minimal	Unique message ID header is now required in every request/response. Correlation message ID header in required for any request via a Hub.