OCPI 2.1.1

Open Charge Point Interface 2.1.1, document version: 2.1.1-RC1

https://github.com/ocpi

15.01.2017

Contents

Version History

Version	Date	Description
2.1.1-RC1	15-01-2017	Fixed the last 4 issues after consulting with the OCPI community.
2.1.1-DRAFT1	30-12-2016	Fixed 3 bugs found in OCPI 2.1, lots of small textual improvements: see changelog
2.1	08-04-2016	Added command module. Added support for real-time authorization. Lots of small improvements: see changelog
2.0-d2	15-02-2016	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	First official release of OCPI.
0.4	04-11-2014	First draft of OCPI. (Also known as Draft v4)
0.3	06-05-2014	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 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 revisions (not the first) will be named: d2, d3, d4 etc.

1 OCPI

1.1 OCPI 2.1.1

During implementation of OCPI 2.1, a number of bugs in the message definition were found.
This forced us to release a bug fix: OCPI 2.1.1.
With the release of OCPI 2.1.1: OCPI 2.1 is deprecated, 2.1 should no longer used and replaced by 2.1.1.
It should be a small effort to upgrade an existing 2.1 implementation to 2.1.1.

1.2 OCPI 2.1.1 - RC1

This document is: Release Candidate 1 (RC1)

All known bugs, typos and textual improvements have been fixed.
A number of high priority issues, not necessarily bugs, but requested by the OCPI community have also been fixed.
For more info on message level changes see changelog.

We want to prevent running into the same problem as we had with OCPI 2.1.

We will make 2.1.1 FINAL after a couple of implementation has successfully tests against each other and no bugs are found.

1.3 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 eVioline 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 eVioline members was defined. This format is currently in use by the majority of the eVioline members. (eVioline 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

- **OCPI** Open Charge Point Interface
- **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 upfront, at least between the cooperating parties.

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

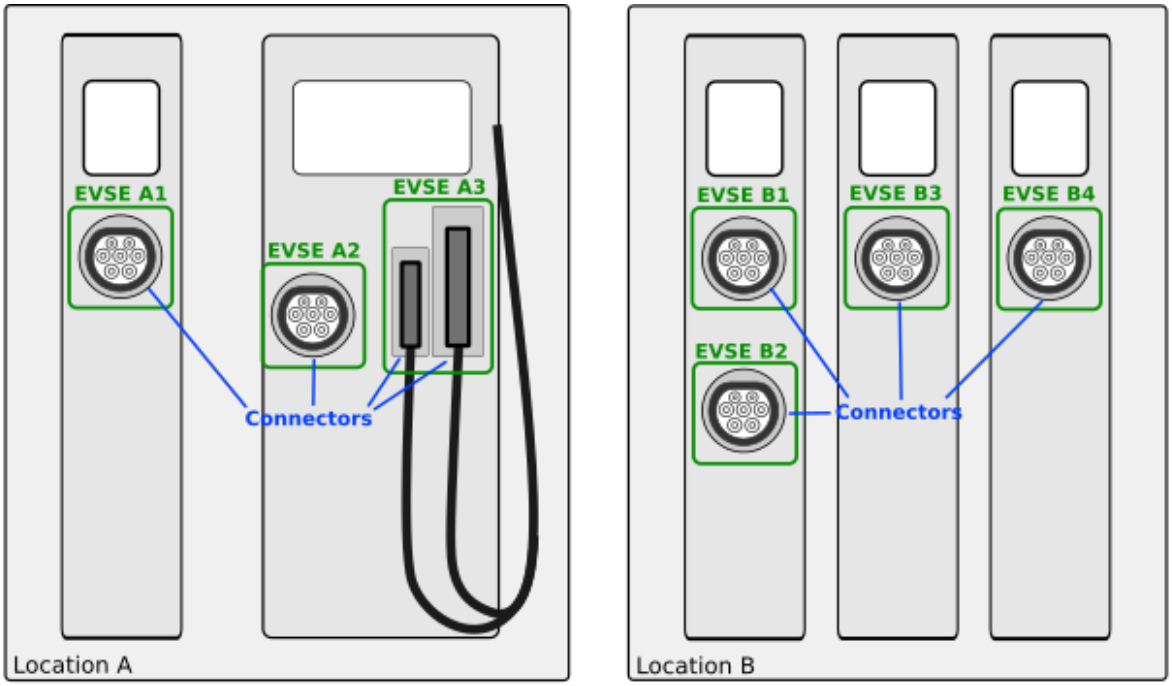
The AFIREV* organisation will keep/keeps the registry for France. It provides operation Id for CPO and eMSP in compliance with eMI3 id structure. The prefix of these Ids is the “fr” country code. AFIREV will also be in charge of the definition of EVSE-Id structure, Charging-Pool-Id structure (location),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



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 during the 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 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).

3.1.3 Request format

Each 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. Its parameter is a string consisting of printable, non-whitespace ASCII characters. The token must uniquely identify the requesting party. This way, the server can use this to link data and commands to this party's account.

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 mimetype of the request body is `application/json` and may contain the data as documented for each endpoint.

3.1.3.1 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.

3.1.3.1.1 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.

3.1.3.1.2 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.

3.1.3.1.3 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 Parameter	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.

3.1.3.1.4 Pagination Examples Example of a required OCPI pagination link header:

```
Link: <https://www.server.com/ocpi/cpo/2.0/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/2.0/cdrs/?offset=200&limit=50>; rel="next"
```

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

```
https://www.server.com/ocpi/cpo/2.0/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.)

```
Link: <https://www.server.com/ocpi/cpo/2.0/cdrs/?offset=100&limit=100&date_from=2016-01-01T00:00:00Z&date_
```

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

```
https://www.server.com/ocpi/cpo/2.0/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 parameter should be set to 100 as well.

```
Link: <https://www.server.com/ocpi/cpo/2.0/cdrs/?offset=100&limit=100>; rel="next"
```

3.1.3.2 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.

3.1.3.3 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.

3.1.4 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/2.0/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.

3.1.4.1 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.

3.1.5 Response format

When a request cannot be accepted, an HTTP error response code is expected including a JSON object that contains more details. HTTP status codes are described on w3.org.

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.
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.

3.1.5.1 Example: Version information response (list of objects)

```
{
    "data": [{
        "version": "1.9",
        "url": "https://example.com/ocpi/cpo/1.9/"
    }, {
        "version": "2.0",
        "url": "https://example.com/ocpi/cpo/2.0/"
    }],
    "status_code": 1000,
    "status_message": "Success",
    "timestamp": "2015-06-30T21:59:59Z"
}
```

3.1.5.2 Example: Version details response (one object)

```
{
    "data": {
        "version": "2.0",
        "endpoints": [{
            "identifier": "credentials",
            "url": "https://example.com/ocpi/cpo/2.0/credentials/"
        }, {
            "identifier": "locations",
            "url": "https://example.com/ocpi/cpo/2.0/locations/"
        }]
    },
    "status_code": 1000,
    "status_message": "Success",
    "timestamp": "2015-06-30T21:59:59Z"
}
```

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

```
{
    "data": {
        "uid": "012345678",
        "type": "RFID",
        "auth_id": "FA54320",
        "visual_number": "DF000-2001-8999",
        "issuer": "TheNewMotion",
        "valid": true,
        "allow_whitelist": true
    },
    "status_code": 1000,
    "status_message": "Success",
    "timestamp": "2015-06-30T21:59:59Z"
}
```

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


```
{
    "data": [{
        "uid": "100012",
        "type": "RFID",
        "auth_id": "FA54320",
        "visual_number": "DF000-2001-8999",
        "issuer": "TheNewMotion",
        "valid": true,
        "allow_whitelist": true
    }, {
        "uid": "100013",
        "type": "RFID",
        "auth_id": "FA543A5",
        "visual_number": "DF000-2001-9000",
        "issuer": "TheNewMotion",
        "valid": true,
        "allow_whitelist": true
    }, {
        "uid": "100014",
        "type": "RFID",
        "auth_id": "FA543BB",
        "visual_number": "DF000-2001-9010",
        "issuer": "TheNewMotion",
        "valid": false,
        "allow_whitelist": true
    }],
    "status_code": 1000,
    "status_message": "Success",
    "timestamp": "2015-06-30T21:59:59Z"
}
```


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


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


3.2 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/2.0/locations` and `/ocpi/emsp/2.0/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/2.0/credentials
Charging location details	locations	https://example.com/ocpi/cpo/2.0/locations

eMSP interface	Identifier	Example URL
Credentials	credentials	https://example.com/ocpi/emsp/2.0/credentials
Charging location updates	locations	https://example.com/ocpi/emsp/2.0/locations

3.3 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
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.

5 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.

Example endpoint structure: `/ocpi/cpo/versions` and `/ocpi/emsp/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 version 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 Data

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

5.1.1 Version *class*

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

5.2 GET

Fetch all supported OCPI versions of this CPO or eMSP.

5.2.1 Example

```
[
    {
        "version": "2.1.1",
        "url": "https://example.com/ocpi/cpo/2.1.1/"
    },
    {
        "version": "2.0",
        "url": "https://example.com/ocpi/cpo/2.0/"
    }
]
```

6 Version details endpoint

Example: `/ocpi/cpo/2.0/` and `/ocpi/emsp/2.0/`

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.

6.1 Data

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

6.1.1 Endpoint *class*

Property	Type	Card.	Description
identifier	ModuleID	1	Endpoint identifier.
url	URL	1	URL to the endpoint.

6.1.2 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	
Commands	commands	
Credentials & registration	credentials	Required for all implementations
Locations	locations	
Sessions	sessions	
Tariffs	tariffs	
Tokens	tokens	

6.1.3 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. (this version)

6.1.3.1 Custom Modules Parties are allowed to create custom modules or customized version of the existing modules.
For this the ModuleID enum can be extended with additional custom moduleIDs.
These custom moduleIDs MAY only be send 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 that they 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`

6.2 GET

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

6.2.1 Example


```
{
    "version": "2.0",
    "endpoints": [
        {
            "identifier": "credentials",
            "url": "https://example.com/ocpi/cpo/2.0/credentials/"
        },
        {
            "identifier": "locations",
            "url": "https://example.com/ocpi/cpo/2.0/locations/"
        }
    ]
}
```

7 Credentials endpoint

Module Identifier: `credentials`

7.1 Interfaces and endpoints

Example: `/ocpi/cpo/2.0/credentials` and `/ocpi/emsp/2.0/credentials`

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

7.1.1 GET Method

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

7.1.2 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 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.

7.1.3 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 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.

7.1.4 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.

7.2 Object description

7.2.1 Credentials object

Property	Type	Card.	Description
token	string(64)	1	The token for the other party to authenticate in your system.
url	URL	1	The URL to your API versions endpoint.
business_details	BusinessDetails	1	Details of the other party.
party_id	string(3)	1	CPO or eMSP ID of this party. (following the 15118 ISO standard).
country_code	string(2)	1	Country code of the country this party is operating in.

The `party_id` and `country_code` are provided here to inform a server about the `party_id` and `country_code` 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 implementation OCPI might push EVSE IDs with a different eMI3 `spot operator`, then the OCPI `party_id` and/or different `country_code`.

7.2.2 Example

```
{
    "url": "https://example.com/ocpi/cpo/",
    "token": "ebf3b399-779f-4497-9b9d-ac6ad3cc44d2",
    "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"
    }
}
```

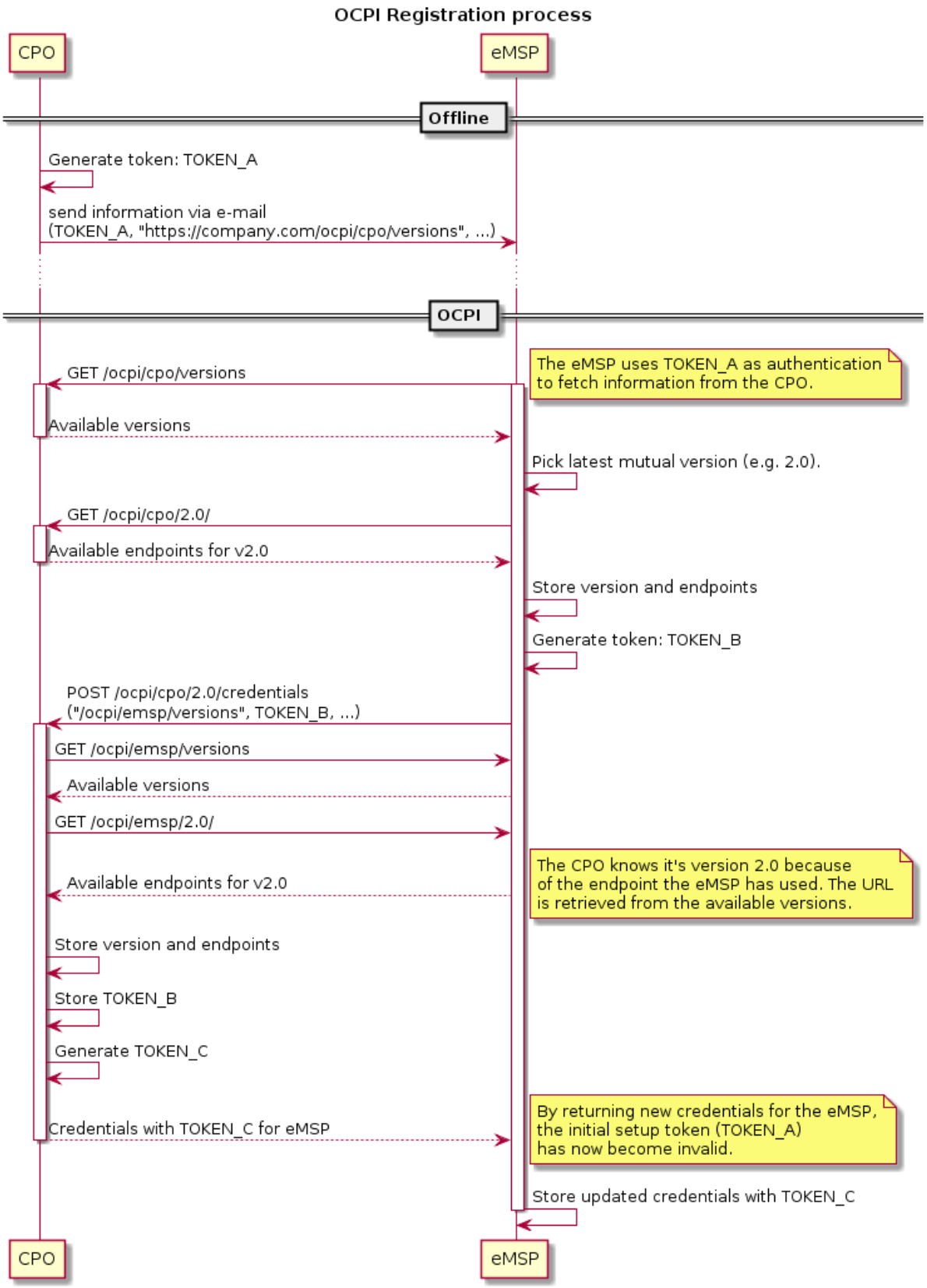
7.3 Use cases

7.3.1 Registration

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

`TOKEN_A` is given offline, after registration store the `TOKEN_C` which will be used in future exchanges.

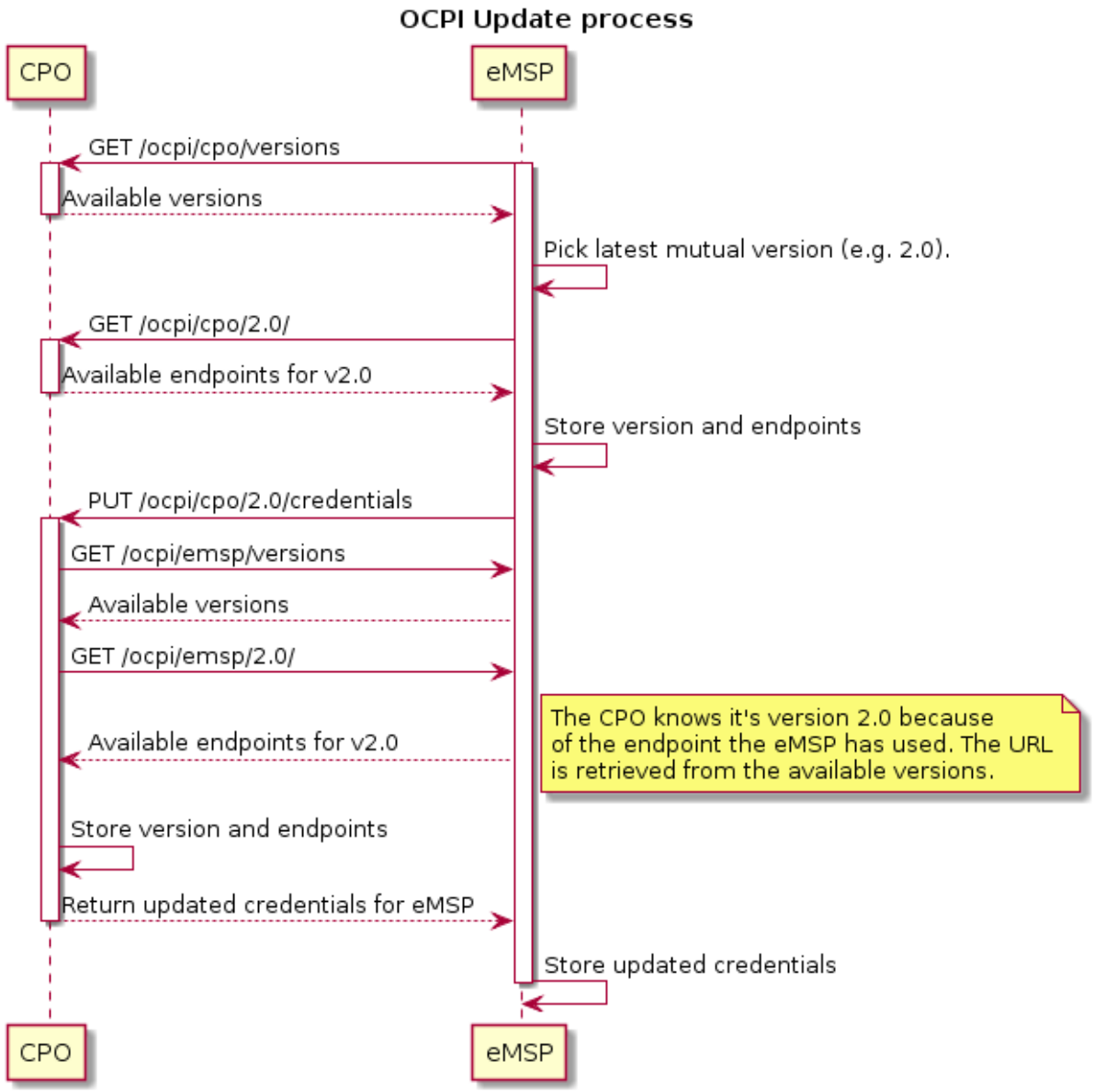
(In the sequence diagrams below we use relative paths as short resource identifiers to illustrate a point; please note that they should really be absolute URLs in any working implementation of OCPI)



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

7.3.2 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.



7.3.3 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.

7.3.4 Updating the credentials and resetting the token

The credentials (or parts thereof, such as the 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.

7.3.5 Errors during registration

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

7.3.6 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.

8 *Locations* module

Module Identifier: `locations`

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

8.1 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.

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.

8.2 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).

8.2.1 CPO Interface

Example endpoint structure: `/ocpi/cpo/2.0/locations`

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

8.2.1.1 GET Method Depending on the URL Segments provided, the GET request can either be used to retrieve information about a list of available locations and EVSEs at this CPO: GET List
Or it can be used to get information about a specific Location, EVSE or Connector: GET Object

8.2.1.1.1 GET List Request Parameters Example endpoint structures for retrieving a list of Locations:

```
/ocpi/cpo/2.0/locations/?date_from=xxx&date_to=yyy
/ocpi/cpo/2.0/locations/?offset=50
/ocpi/cpo/2.0/locations/?limit=100
/ocpi/cpo/2.0/locations/?offset=50&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.

8.2.1.1.2 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.

8.2.1.1.3 GET Object Request Parameters Example endpoint structures for a specific Location, EVSE or Connector:
`/ocpi/cpo/2.0/locations/{location_id}`
`/ocpi/cpo/2.0/locations/{location_id}/{evse_uid}`
`/ocpi/cpo/2.0/locations/{location_id}/{evse_uid}/{connector_id}`

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
location_id	string(39)	yes	Location.id of the Location object to retrieve.
evse_uid	string(39)	no	Evse.uid, required when requesting an EVSE or Connector object.
connector_id	string(36)	no	Connector.id, required when requesting a Connector object.

8.2.1.1.4 GET Object Response Data The response contains the requested object.

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

8.2.2 eMSP Interface

Locations is a client owned object, so the end-points need to contain the required extra fields: {party_id} and {country_code}.
Example endpoint structures:
`/ocpi/emsp/2.0/locations/{country_code}/{party_id}/{location_id}`
`/ocpi/emsp/2.0/locations/{country_code}/{party_id}/{location_id}/{evse_uid}`
`/ocpi/emsp/2.0/locations/{country_code}/{party_id}/{location_id}/{evse_uid}/{connector_id}`

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)*

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

8.2.2.1.1 Request Parameters The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the CPO requesting this PUT to the eMSP system.
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
location_id	string(39)	yes	Location.id of the Location object to retrieve.
evse_uid	string(39)	no	Evse.uid, required when requesting an EVSE or Connector object.
connector_id	string(36)	no	Connector.id, required when requesting a Connector object.

8.2.2.1.2 Response Data The response contains the requested object.

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

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

8.2.2.2.1 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	string(2)	yes	Country code of the CPO requesting this PUT to the eMSP system.
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
location_id	string(39)	yes	Location.id of the new Location object, or the Location of which an EVSE or Location object is send
evse_uid	string(39)	no	Evse.uid, required when an EVSE or Connector object is send/replaced.
connector_id	string(36)	no	Connector.id, required when a Connector object is send/replaced.

8.2.2.2.2 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.
> EVSE	1	New EVSE object, or EVSE object to replace.
> Connector	1	New Connector object, or Connector object to replace.

8.2.2.3 PATCH Method Same as the PUT method, but only the fields/objects that have to be updated have to be present, other fields/objects that are not specified are considered unchanged.

8.2.2.3.1 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/2.0/locations/NL/TNM/1012/3255

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

8.2.2.3.2 Example: change the location name In this example the name of location 1012 is updated.

PATCH To URL: https://www.server.com/ocpi/emsp/2.0/locations/NL/TNM/1012

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

8.2.2.3.3 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/2.0/locations/NL/TNM/1012/3255/2

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

8.2.2.3.4 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/2.0/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
}
```


8.2.2.3.5 Example: delete an EVSE An EVSE can be deleted by updating its *status* property.

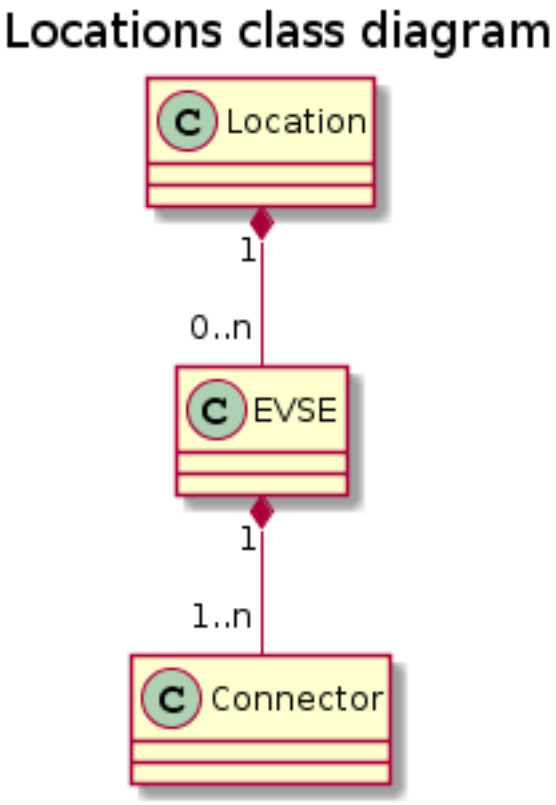
PATCH To URL: https://www.server.com/ocpi/emsp/2.0/locations/NL/TNM/1012/3256


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


*Note: To inform that an EVSE is scheduled for removal, the status*schedule field can be used._

8.3 Object description

Location, EVSE and Connector have the following relation.



8.3.1 *Location* Object

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

Property	Type	Card.	Description
id	string(39)	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)	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.
related_locations	AdditionalGeoLocation	*	Geographical location of related points relevant to the user.
evses	EVSE	*	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 `api_info` endpoint 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).

8.3.1.1 Example


```
{
    "id": "LOC1",
    "type": "ON_STREET",
    "name": "Gent Zuid",
    "address": "F.Rooseveltlaan 3A",
    "city": "Gent",
    "postal_code": "9000",
    "country": "BEL",
    "coordinates": {
        "latitude": "51.04759",
        "longitude": "3.72994"
    },
    "evses": [{
        "uid": "3256",
        "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",
        "id": "BE-BEC-E041503002",
        "status": "RESERVED",
        "capabilities": [
            "RESERVABLE"
        ],
        "connectors": [{
            "id": "1",
            "status": "RESERVED",
            "standard": "IEC_62196_T2",
            "format": "SOCKET",
            "power_type": "AC_3_PHASE",
            "voltage": 220,
            "amperage": 16,
            "tariff_id": "12"
        }],
        "physical_reference": "2",
        "floor_level": "-2",
        "last_updated": "2015-06-29T20:39:09Z"
    }],
    "operator": {
        "name": "BeCharged"
    },
    "last_updated": "2015-06-29T20:39:09Z"
}
```

8.3.2 *EVSE* Object

The *EVSE* object describes the part that controls the power supply to a single EV in a single session. It always belongs to a *Location* object. 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	string(39)	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	string(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).

8.3.3 *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	string(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].
tariff_id	string(36)	?	Identifier of the current charging tariff structure. 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).

8.4 Data types

8.4.1 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]{6}`
longitude	string(11)	1	Longitude of the point in decimal degree. Example: -126.104965. Decimal separator: “.” Regex: `-?[0-9]{1,3}\.[0-9]{6}`
name	DisplayText	?	Name of the point in local language or as written at the location. For example the street name of a parking lot entrance or it’s number.

8.4.2 BusinessDetails *class*

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

8.4.3 Capability *enum*

The capabilities of an EVSE.

Value	Description
CHARGING_PROFILE_CAPABLE	The EVSE supports charging profiles. Sending Charging Profiles is not yet supported by OCPI.
CREDIT_CARD_PAYABLE	Charging at this EVSE can be payed with credit 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
UNLOCK_CAPABLE	Connectors have mechanical lock that can be requested by the eMSP to be unlocked.

8.4.4 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.

8.4.5 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)

8.4.6 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.)*

8.4.6.1 Examples

8.4.6.1.1 Simple:

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

8.4.6.1.2 Tariff name based:

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

8.4.6.1.3 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": 0.00006, },
            { "source": "CARBON_DIOXIDE", "amount": 372,     }
        ],
    "supplier_name":       "E.ON Energy Deutschland",
    "energy_product_name": "E.ON DirektStrom eco"
    }
```

8.4.7 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.

8.4.8 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.

8.4.9 EnvironmentalImpact *class*

Key-value pairs (enum + amount) of waste and carbon dioxide emittion per kWh.

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

8.4.10 EnvironmentalImpactCategory *enum*

Categories of environmental impact values.

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

8.4.11 ExceptionalPeriod *class*

Specifies one exceptional period for opening or access hours.

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

8.4.12 Facility *enum*

Value	Description
HOTEL	A hotel.
RESTAURANT	A restaurant.
CAFE	A cafe.
MALL	A mall or shopping center.
SUPERMARKET	A supermarket.
SPORT	Sport facilities: gym, field etc.
RECREATION_AREA	A Recreation area.
NATURE	Located in, or close to, a park, nature reserve/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.

8.4.13 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]{6}`
longitude	string(11)	1	Longitude of the point in decimal degree. Example: -126.104965. Decimal separator: “.” Regex: `-?[0-9]{1,3}\.[0-9]{6}`

8.4.14 Hours *class*

Opening and access hours of the location.

Field Name	Field Type	Card.	Description
Choice: one of two			
> regular_hours	RegularHours	*	Regular hours, weekday based. Should not be set for representing 24/7 as this is the most common case.
> twentyfourseven	boolean	1	True to represent 24 hours a day and 7 days a week, except the given exceptions.
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.

8.4.15 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.

Field Name	Field 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

8.4.16 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

8.4.17 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).

8.4.18 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.

8.4.19 PowerType *enum*

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

8.4.20 RegularHours *class*

Regular recurring operation or access hours

Field Name	Field 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.

8.4.20.1 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": "2014-06-21T09:00:00+02:00",
      "period_end": "2014-06-21T12:00:00+02:00"
    }
  ],
  "exceptional_closings": [
    {
      "period_begin": "2014-06-24T00:00:00+02:00",
      "period_end": "2014-06-25T00:00:00+02:00"
    }
  ]
}
```

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.

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

8.4.21 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).
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)

8.4.22 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.

9 *Sessions* module

Module Identifier: `sessions`

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

9.1 Flow and Lifecycle

9.1.1 Push model

When the CPO creates a Session object they push it to the 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.

9.1.2 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.

9.2 Interfaces and endpoints

9.2.1 CPO Interface

Example endpoint structure: `/ocpi/cpo/2.0/sessions/?date_from=xxx&date_to=yyy`

Method	Description
GET	Fetch Session objects of charging sessions last updated 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 Sessions from the CPO systems.

9.2.1.1.1 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.

9.2.1.1.2 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

9.2.2 eMSP Interface

Sessions is a client owned object, so the end-points need to contain the required extra fields: {party_id} and {country_code}.
Example endpoint structure:
`/ocpi/emsp/2.0/sessions/{country_code}/{party_id}/{session_id}`

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

9.2.2.1 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.

9.2.2.1.1 Request Parameters The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the CPO requesting this GET to the eMSP system.
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this GET to the eMSP system.
session_id	string(36)	yes	id of the Session object to get from the eMSP system.

9.2.2.1.2 Response Data The response contains the request Session object, if available.

Datatype	Card.	Description
Session	1	Session object requested.

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

9.2.2.2.1 Request Body The request contains the new or updated Session object.

Type	Card.	Description
Session	1	new Session object.

9.2.2.2.2 Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the CPO requesting this PUT to the eMSP system.
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
session_id	string(36)	yes	id of the new or updated Session object.

9.2.2.3 PATCH Method

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

9.2.2.3.1 Example: update the total cost

```
PATCH To URL: https://www.server.com/ocpi/cpo/2.0/sessions/NL/TNM/101
```

```
{
    "total_cost": 0.60
}
```

9.3 Object description

9.3.1 *Session* Object

Property	Type	Card.	Description
id	string(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.
auth_id	string(36)	1	Reference to a token, identified by the auth_id field of the Token.
auth_method	AuthMethod	1	Method used for authentication.
location	Location	1	The location where this session took place, including only the relevant EVSE and connector
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	number	?	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).

9.3.1.1 Examples

9.3.1.2 Simple Session example of a just starting session

```
{
    "id": "101",
    "start_datetime": "2015-06-29T22:39:09Z",
    "kwh": 0.00,
    "auth_id": "DE8ACC12E46L89",
```

```
    "location": {
        "id": "LOC1",
        "type": "on_street",
        "name": "Gent Zuid",
        "address": "F.Rooseveltlaan 3A",
        "city": "Gent",
        "postal_code": "9000",
        "country": "BE",
        "coordinates": {
            "latitude": "3.72994",
            "longitude": "51.04759"
        },
        "evse": {
            "uid": "3256",
            "evse_id": "BE-BEC-E041503003",
            "STATUS": "AVAILABLE",
            "connectors": [{
                "id": "1",
                "standard": "IEC_62196_T2",
                "format": "SOCKET",
                "power_type": "AC_1_PHASE",
                "voltage": 230,
                "amperage": 64,
                "tariff_id": "11",
                "last_updated": "2015-06-29T22:39:09Z"
            }],
            "last_updated": "2015-06-29T22:39:09Z"
        },
        "last_updated": "2015-06-29T22:39:09Z"
    },
    "currency": "EUR",
    "total_cost": 2.50,
    "status": "PENDING",
    "last_updated": "2015-06-29T22:39:09Z"
}
```

9.3.1.2.1 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,
    "auth_id": "DE8ACC12E46L89",
    "location": {
        "id": "LOC1",
        "type": "on_street",
        "name": "Gent Zuid",
        "address": "F.Rooseveltlaan 3A",
        "city": "Gent",
        "postal_code": "9000",
        "country": "BE",
        "coordinates": {
            "latitude": "3.72994",
            "longitude": "51.04759"
        },
        "evse": {
            "uid": "3256",
            "evse_id": "BE-BEC-E041503003",
            "STATUS": "AVAILABLE",
            "connectors": [{
                "id": "1",
                "standard": "IEC_62196_T2",
                "format": "SOCKET",
                "power_type": "AC_1_PHASE",
                "voltage": 230,
                "amperage": 64,
                "tariff_id": "11",
                "last_updated": "2015-06-29T23:09:10Z"
            }],
            "last_updated": "2015-06-29T23:09:10Z"
        },
        "last_updated": "2015-06-29T23:09:10Z"
```

```
    },
    "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
        }]
    }],
    "total_cost": 8.50,
    "status": "COMPLETED",
    "last_updated": "2015-06-29T23:09:10Z"
}
```

9.4 Data types

Describe all datatypes used in this object

9.4.1 SessionStatus *enum*

Property	Description
ACTIVE	The session is accepted and active.
COMPLETED	The session is finished successfully.
INVALID	The session is declared invalid and will not be billed.
PENDING	The session is pending and has not yet started. This is the initial state.

.

10 *CDRs* module

Module Identifier: `cdrs`

A **Charge Detail Record** is the description of a concluded charging
session. The CDR is the only billing-relevant object.
CDRs are send 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
ASAP. But if there is an agreement between parties to send them for example once a month, that is also allowed by OCPI.

10.1 Flow and Lifecycle

CDRs are created by the CPO. They 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.

10.1.1 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.

10.1.2 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.

10.2 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.

10.2.1 CPO Interface

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

Example endpoint structure: `/ocpi/cpo/2.0/cdrs/?date_from=xxx&date_to=yyy`

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

10.2.1.1 GET Method

Fetch CDRs from the CPO systems.

10.2.1.1.1 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.

10.2.1.1.2 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.

10.2.2 eMSP Interface

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

Example endpoint structure: `/ocpi/emsp/2.0/cdrs`

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

10.2.2.1 GET Method

Fetch CDRs from the eMSP system.

10.2.2.1.1 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.

10.2.2.1.2 Response Data

The endpoint returns the requested CDR, if it exists

Datatype	Card.	Description
CDR	1	Requested CDR object.

10.2.2.2 POST Method Creates a new CDR.

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

10.2.2.2.1 Request Body In the post request the new CDR object is sent.

Type	Card.	Description
CDR	1	New CDR object.

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

10.2.2.2.2 Response Headers Example: Location: /ocpi/emsp/2.0/cdrs/123456

10.3 Object description

10.3.1 *CDR* Object

The *CDR* object describes the Charging Session and its costs. How these costs are build up etc.

Property	Type	Card.	Description
id	CiString(36)	1	Uniquely identifies the CDR within the CPOs platform (and suboperator platforms).
start_date_time	DateTime	1	Start timestamp of the charging session.
stop_date_time	DateTime	1	Stop timestamp of the charging session.
auth_id	string(36)	1	Reference to a token, identified by the auth_id field of the Token.
auth_method	AuthMethod	1	Method used for authentication.
location	Location	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 charging_periods	Tariff ChargingPeriod	*	List of relevant tariff elements, see: Tariffs. When relevant, a “Free of Charge” tariff should also be in this list, and point to a defined “Free of Charge” tariff. 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	number	1	Total cost of this transaction.
total_energy	number	1	Total energy charged, in kWh.
total_time	number	1	Total time charging, in hours.
total_parking_time	number	?	Total time not charging, 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.
last_updated	DateTime	1	Timestamp when this CDR was last updated (or created).

10.3.1.1 Example of a CDR


```
{
    "id": "12345",
    "start_date_time": "2015-06-29T21:39:09Z",
    "stop_date_time": "2015-06-29T23:37:32Z",
    "auth_id": "DE8ACC12E46L89",
    "auth_method": "WHITELIST",
    "location": {
        "id": "LOC1",
        "type": "on_street",
        "name": "Gent Zuid",
        "address": "F.Rooseveltlaan 3A",
        "city": "Gent",
        "postal_code": "9000",
        "country": "BE",
        "coordinates": {
            "latitude": "3.72994",
            "longitude": "51.04759"
        },
        "evse": {
            "uid": "3256",
            "evse_id": "BE-BEC-E041503003",
            "STATUS": "AVAILABLE",
            "connectors": [{
                "id": "1",
                "standard": "IEC-62196-T2",
                "format": "SOCKET",
                "power_type": "AC_1_PHASE",
                "voltage": 230,
                "amperage": 64,
                "tariff_id": "11",
                "last_updated": "2015-06-29T21:39:01Z"
            }],
            "last_updated": "2015-06-29T21:39:01Z"
        },
        "last_updated": "2015-06-29T21:39:01Z"
    },
    "currency": "EUR",
    "tariffs": [{
        "id": "12",
        "currency": "EUR",
        "elements": [{
            "price_components": [{
                "type": "TIME",
                "price": "2.00",
                "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
        }]
    }],
    "total_cost": 4,00,
    "total_usage": [{
        "type": "TIME",
        "volume": 1.973
    }, {
        "type": "ENERGY",
        "volume": 15.342
    }],
    "last_updated": "2015-06-29T22:01:13Z"
}
```

10.4 Data types

10.4.1 AuthMethod *enum*

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

10.4.2 CdrDimension *class*

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

10.4.3 CdrDimensionType *enum*

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.

10.4.4 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.

11 *Tariffs* module

Module Identifier: `tariffs`

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

11.1 Flow and Lifecycle

11.1.1 Push model

When the CPO creates a new Tariff they push them to the eMSPs by calling the PUT 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.

11.1.2 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.

11.2 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.

11.2.1 CPO Interface

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

Example endpoint structure: `/ocpi/cpo/2.0/tariffs/?date_from=xxx&date_to=yyy`

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

11.2.1.1 GET Method Fetch information about all Tariffs.

11.2.1.1.1 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.

11.2.1.1.2 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.

11.2.2 eMSP Interface

Tariffs is a client owned object, so the end-points need to contain the required extra fields: {party_id} and {country_code}.
Example endpoint structure:
`/ocpi/emsp/2.0/tariffs/{country_code}/{party_id}/{tariff_id}`

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.
DELETE	Remove Tariff object which is no longer valid

11.2.2.1 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.

11.2.2.1.1 Request Parameters The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the CPO requesting this PUT to the eMSP system.
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
tariff_id	string(36)	yes	Tariff.id of the Tariff object to retrieve.

11.2.2.1.2 Response Data The response contains the requested object.

Type	Card.	Description
Tariff	1	The requested Tariff object.

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

11.2.2.2.1 Request Body In the put request the new or updated Tariff object is sent.

Type	Card.	Description
Tariff	1	New or updated Tariff object

11.2.2.2.2 Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the CPO requesting this PUT to the eMSP system.
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
tariff_id	string(36)	yes	Tariff.id of the (new) Tariff object (to replace).

11.2.2.2.3 Example: New Tariff 2 euro per hour

PUT To URL: https://www.server.com/ocpi/emsp/2.0/tariffs/NL/TNM/12

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

11.2.2.3 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.

11.2.2.3.1 Example: Change Tariff to 2,50

PUT To URL: https://www.server.com/ocpi/emsp/2.0/tariffs/NL/TNM/12

```
{
    "elements": [{
        "price_components": [{
            "type": "TIME",
            "price": 2.50,
            "step_size": 300
        }]
    }]
}
```

11.2.2.4 DELETE Method

Delete a no longer valid Tariff object.

11.2.2.4.1 Request Parameters

The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the CPO requesting this PUT to the eMSP system.
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
tariff_id	string(36)	yes	Tariff.id of the Tariff object to delete.

11.3 Object description

11.3.1 *Tariff* Object

A Tariff Object consists of a list of one or more 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	string(36)	1	Uniquely identifies the tariff within the CPOs platform (and suboperator platforms).
currency	string(3)	1	Currency of this tariff, ISO 4217 Code
tariff_alt_text	DisplayText	*	List of multi language alternative tariff info text
tariff_alt_url	URL	?	Alternative URL to tariff info
elements	TariffElement	+	List of tariff elements
energy_mix	EnergyMix	?	Details on the energy supplied with this tariff.
last_updated	DateTime	1	Timestamp when this Tariff was last updated (or created).

11.3.1.1 Examples

11.3.1.1.1 Simple Tariff example 2 euro per hour

```
{
    "id": "12",
    "currency": "EUR",
    "elements": [{
        "price_components": [{
            "type": "TIME",
            "price": 2.00,
            "step_size": 300
        }]
    }],
    "last_updated": "2015-06-29T20:39:09Z"
}
```

11.3.1.1.2 Simple Tariff example with alternative multi language text

```
{
    "id": "12",
    "currency": "EUR",
    "tariff_alt_text": [{
        "language": "en",
        "text": "2 euro p/hour"
    }, {
        "language": "nl",
        "text": "2 euro p/uur"
    }],
    "elements": [{
        "price_components": [{
            "type": "TIME",
            "price": 2.00,
            "step_size": 300
        }]
    }],
    "last_updated": "2015-06-29T20:39:09Z"
}
```

11.3.1.1.3 Simple Tariff example with alternative URL

```
{
    "id": "12",
    "currency": "EUR",
    "tariff_alt_url": "https://company.com/tariffs/12",
    "elements": [{
        "price_components": [{
            "type": "TIME",
            "price": 2.00,
            "step_size": 300
        }]
    }],
    "last_updated": "2015-06-29T20:39:09Z"
}
```

11.3.1.1.4 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)

```
{
    "id": "11",
    "currency": "EUR",
    "tariff_alt_url": "https://company.com/tariffs/11",
    "elements": [{
        "price_components": [{
            "type": "FLAT",
            "price": 2.50,
            "step_size": 1
        }]
    }, {
        "price_components": [{
            "type": "TIME",
            "price": 1.00,
            "step_size": 900
        }],
        "restrictions": {
            "max_power": 32.00
        }
    }, {
        "price_components": [{
            "type": "TIME",
            "price": 2.00,
            "step_size": 600
        }],
        "restrictions": {
            "min_power": 32.00,
            "day_of_week": ["MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"]
        }
    }, {
        "price_components": [{
            "type": "TIME",
            "price": 1.25,
            "step_size": 600
        }],
        "restrictions": {
            "min_power": 32.00,
            "day_of_week": ["SATURDAY", "SUNDAY"]
        }
    }, {
        "price_components": [{
            "type": "PARKING_TIME",
            "price": 5.00,
            "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,
            "step_size": 300
        }],
        "restrictions": {
            "start_time": "10:00",
            "end_time": "17:00",
            "day_of_week": ["SATURDAY"]
        }
    }],
    "last_updated": "2015-06-29T20:39:09Z"
}
```

11.3.1.1.5 Free of Charge Tariff example

```
{
    "id": "12",
    "currency": "EUR",
    "elements": [{
        "price_components": [{
            "type": "FLAT",
            "price": 0.00,
            "step_size": 0
        }]
    }],
    "last_updated": "2015-06-29T20:39:09Z"
}
```

11.4 Data types

11.4.1 DayOfWeek *enum*

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

11.4.2 PriceComponent *class*

Property	Type	Card.	Description
type	TariffDimensionType	1	Type of tariff dimension
price	number	1	price per unit for this tariff dimension
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.

The step_size also depends on the type, every type (except FLAT) defines a step_size multiplier. this is the size of every 'step' and the unit.
For example: PARKING_TIME has 'step_size multiplier: 1 second' That means that the step_size of a PriceComponent is muliplied by 1 second.
Thus a step_size = 300 means 300 seconds.

11.4.3 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

11.4.4 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

11.4.5 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

.

12 *Tokens* module

Module Identifier: `tokens`

The tokens module gives CPOs knowledge of the token information of an eMSP.
eMSPs can push Token information to CPOs, CPOs can build a cache of known Tokens.
When a request to authorize comes from a Charge Point, the CPO can check against this cache.
With this cached information they know to which eMSP they can later send a CDR.

12.1 Flow and Lifecycle

12.1.1 Push model

When the MSP creates a new Token object they push it to the CPO by calling PUT on the CPOs Tokens endpoint with the newly created Token object.

Any changes to Token in the eMSP system are send to the CPO system by calling, either the PUT or the PATCH on the CPOs 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 CPOs Tokens endpoint with the updated Token.

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

12.1.2 Pull model

When a CPO is not sure about the state of the list of known Tokens, or wants to request the full list as a start-up of their system, the CPO can call the GET on the eMSPs 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.

12.1.3 Real-time authorization

An eMSP might want their Tokens to be authorization '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`.

12.2 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.

12.2.1 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}.
Example endpoint structure:
`/ocpi/cpo/2.0/tokens/{country_code}/{party_id}/{token_uid}`

Method	Description
GET	Retrieve a Token as it is stored in the CPO system.
POST	n/a
PUT	Push new/updated Token object to the CPO.
PATCH	Notify the CPO of partial updates to a Token.
DELETE	n/a, (Use PUT, Tokens cannot be removed).

12.2.1.1 GET Method If the eMSP wants to check the status of a Token in the CPO system it might GET the object from the CPO system for validation purposes. The eMSP is the owner of the objects, so it would be illogical if the CPO system had a different status or was missing an object.

12.2.1.1.1 Request Parameters The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the eMSP requesting this GET from the CPO system.
party_id	string(3)	yes	Party ID (Provider ID) of the eMSP requesting this GET from the CPO system.
token_uid	string(36)	yes	Token.uid of the Token object to retrieve.

12.2.1.1.2 Response Data The response contains the requested object.

Type	Card.	Description
Token	1	The requested Token object.

12.2.1.2 PUT Method New or updated Token objects are pushed from the eMSP to the CPO.

12.2.1.2.1 Request Body In the put request a the new or updated Token object is send.

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

12.2.1.2.2 Request Parameters The following parameters can be provided as URL segments.

Parameter	Datatype	Required	Description
country_code	string(2)	yes	Country code of the eMSP sending this PUT request to the CPO system.
party_id	string(3)	yes	Party ID (Provider ID) of the eMSP sending this PUT request to the CPO system.
token_uid	string(36)	yes	Token.uid of the (new) Token object (to replace).

12.2.1.2.3 Example: put a new Token

PUT To URL: https://www.server.com/ocpi/cpo/2.0/tokens/NL/TNM/012345678


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

12.2.1.3 PATCH Method Same as the PUT method, but only the fields/objects that have to be updated have to be present, other fields/objects that are not specified are considered unchanged.

12.2.1.3.1 Example: invalidate a Token

PATCH To URL: https://www.server.com/ocpi/cpo/2.0/tokens/NL/TNM/012345678

```
{
  "valid": false
}
```

12.2.2 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.

Example endpoint structure: `/ocpi/emsp/2.0/tokens/?date_from=xxx&date_to=yyy`

Method	Description
GET	Get the list of known Tokens, last updated between the {date_from} and {date_to} (paginated)
POST	Real-time authorization request
PUT	n/a
PATCH	n/a
DELETE	n/a

12.2.2.1 GET Method Fetch information about Tokens known in the eMSP systems.

12.2.2.1.1 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.

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.

12.2.2.1.2 Response Data The endpoint response with list of valid Token objects, the header will contain the pagination related headers.

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

Type	Card.	Description
Token	*	List of all tokens.

12.2.2.2 POST Method Do a 'real-time' authorization request to the eMSP system, validating if a Token might be used (at the optionally given Location).

Example endpoint structure:
`/ocpi/emsp/2.0/tokens/{token_uid}/authorize`
The `/authorize` is required for the real-time authorize request.

When the eMSP receives a 'real-time' authorization request from a CPO that contains to little information (no LocationReferences provided) to determine if the Token might be used, the eMSP SHOULD respond with the OCPI status: 2002

12.2.2.2.1 Request Parameters

The following parameter has to be provided as URL segments.

Parameter	Datatype	Required	Description
token_uid	string(36)	yes	Token.uid of the Token for which this authorization is.

12.2.2.2.2 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.

12.2.2.2.3 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.

12.3 Object description

12.3.1 *AuthorizationInfo* Object

Property	Type	Card.	Description
allowed	Allowed	1	Status of the Token, and if it is allowed to charge at the optionally given location.
location	LocationReferences	?	Optional reference to the location if it was request 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.

12.3.2 *Token* Object

Property	Type	Card.	Description
uid	string(36)	1	Identification used by CPO system to identify this token. Currently, in most cases, this is the RFID hidden ID as read by the RFID reader.
type	TokenType	1	Type of the token
auth_id	string(36)	1	Uniquely identifies the EV Driver contract token within the eMSPs 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 auth_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,
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.
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 an eMSPs system.

12.3.2.1 Example

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

12.4 Data types

12.4.1 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.

12.4.2 LocationReferences *class*

References to location details.

Field Name	Field Type	Card.	Description
location_id	string(39)	1	Uniquely identifier for the location.
evse_uids	string(39)	*	Uniquely identifier for EVSEs within the CPOs platform for the EVSE within the the given location.
connector_ids	string(36)	*	Identifies the connectors within the given EVSEs.

12.4.3 TokenType *enum*

Value	Description
OTHER	Other type of token
RFID	RFID Token

12.4.4 WhitelistType *enum*

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

Value	Description
ALWAYS	Token always has to whitelisted, realtime authorization is not possible/allowed.
ALLOWED	It is allowed to whitelist the token, realtime authorization is also allowed.
ALLOWED_OFFLINE	Whitelisting is only allowed when CPO cannot reach the eMSP (communication between CPO and eMSP is offline)
NEVER	Whitelisting is never allowed/forbidden, only realtime authorization allowed. Token should always be authorized by the eMSP.

13 *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`
- `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

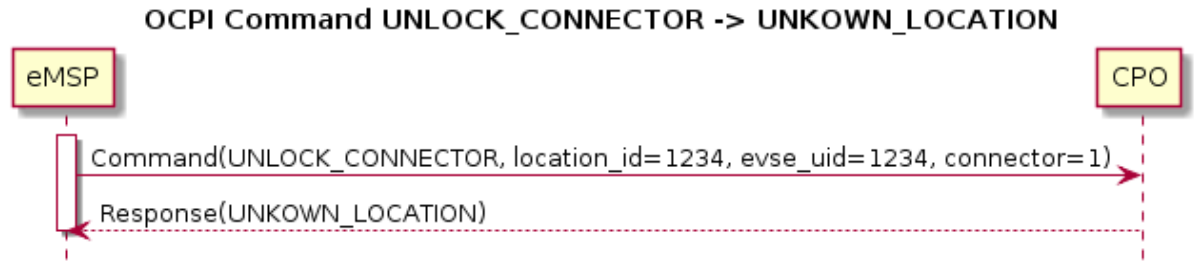
13.1 Flow

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.

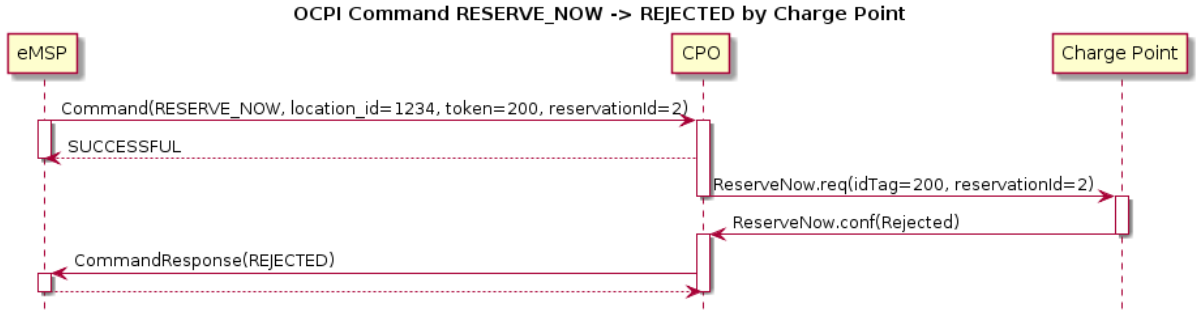
The eMSP send a request to a CPO, via the CPO Commands interface. The CPO checks if it can send the request to a Charge Point and will respond to the request with a status, indicating if the request can be sent to a Charge Point.

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 mean that the command was executed successfully. The CPO will forward this command in a new POST request to the eMSP Commands interface.

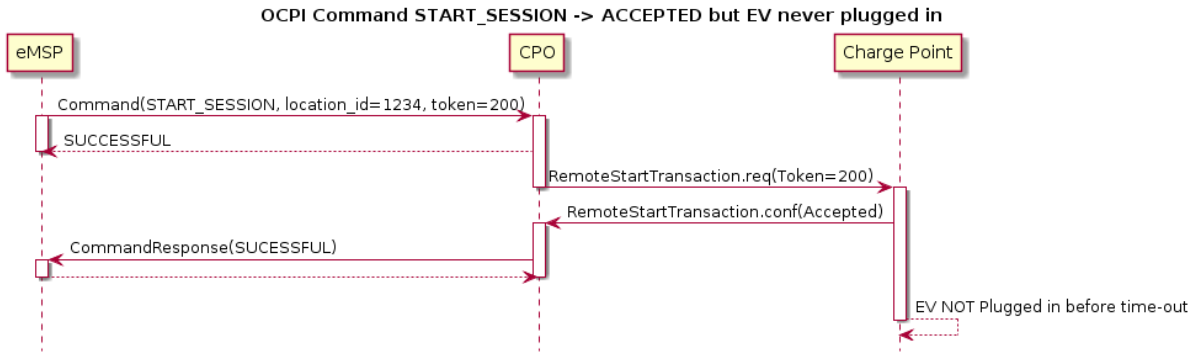
The following examples try to give insight into the message flow and the asynchronous nature of the OCPI Commands.
Example of a `UNLOCK_CONNECTOR` that fails because the Location is not known by the CPO.



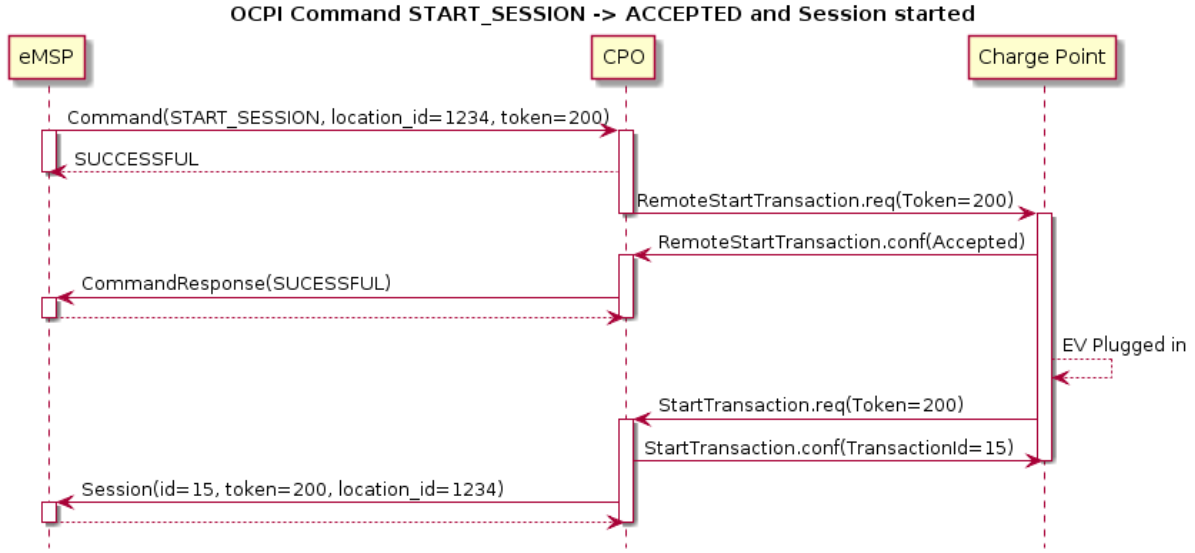
Example of a `RESERVE_NOW` that is rejected by the Charge Point.



Example of a `START_SESSION` that is accepted, but no new Session is started because EV not plugged in before end of time-out.



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



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

13.2 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 Charge Point and an eMSP interface to receive the response from the Charge Point asynchronously.

13.2.1 CPO Interface

Example endpoint structure: `/ocpi/cpo/2.0/commands/{command}`

Method	Description
GET	n/a
POST	Send a command to the CPO, requesting the CPO to send the command to the Charge Point
PUT	n/a
PATCH	n/a
DELETE	n/a

13.2.1.1 POST Method

13.2.1.1.1 Request Parameters The following parameters can be provided as URL segments.

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

13.2.1.2 Request Body Depending on the `command` parameter the body SHALL contain the applicable object for that command.

Type	Card.	Description
Choice: one of four		
> 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.

13.2.1.2.1 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
CommandResponseType	1	Result of the command request by the CPO (not the Charge Point).

13.2.2 eMSP Interface

The eMSP interface receives the asynchronous responses.

Example endpoint structure:
`/ocpi/emsp/2.0/commands/{command}`
`/ocpi/emsp/2.0/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

13.2.2.1 POST Method

13.2.2.1.1 Request Parameters There are no URL segment parameters required by OCPI.
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 simultanous commands, for example by adding a unique id as a URL segment.

Example:
`/ocpi/emsp/2.0/commands/RESERVE_NOW/1234`
`/ocpi/emsp/2.0/commands/UNLOCK_CONNECTOR/2`

Datatype	Card.	Description
CommandResponseType	1	Result of the command request by the CPO (not the Charge Point).

13.2.2.2 Request Body

13.3 Object description

13.3.1 *CommandResponse* Object

Property	Type	Card.	Description
result	CommandResponseType	1	Result of the command request as sent by the Charge Point to the CPO.

13.3.2 *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	string(39)	1	Location.id of the Location (belonging to the CPO this request is send to) for which to reserve an EVSE.
evse_uid	string(39)	?	Optional EVSE.uid of the EVSE of this Location if a specific EVSE has to be reserved.

13.3.3 *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	string(39)	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	string(39)	?	Optional EVSE.uid of the EVSE of this Location on which a session is to be started.

13.3.4 *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	string(36)	1	Session.id of the Session that is requested to be stopped.

13.3.5 *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	string(39)	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	string(39)	1	EVSE.uid of the EVSE of this Location of which it is requested to unlock the connector.
connector_id	string(36)	1	Connector.id of the Connector of this Location of which it is requested to unlock.

13.4 Data types

13.4.1 CommandResponseType *enum*

The command requested.

Value	Description
NOT_SUPPORTED	The requested command is not supported by this CPO, Charge Point, EVSE etc.
REJECTED	Command request rejected by the CPO or Charge Point.
ACCEPTED	Command request accepted by the CPO or Charge Point.
TIMEOUT	Command request timeout, no response received from the Charge Point in an reasonable time.
UNKNOWN_SESSION	The Session in the requested command is not known by this CPO.

13.4.2 CommandType *enum*

The command requested.

Value	Description
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 of the eMSP. This command SHALL never be allowed to be sent directly by the EV-Driver.
The `UNLOCK_CONNECTOR` is intended to be used in the rare situation that the connector is not unlocked successfully after a transaction is stopped. The mechanical unlock of the lock mechanism might get stuck, for example: fail when there is tension on the charging cable when the Charge Point tries to unlock the connector.
In such a situation the EV-Driver can call either the CPO or the eMSP to retry the unlocking.

14 Types

14.1 CiString *type*

Case Insensitive String. Only printable ASCII allowed.

14.2 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:

```
2015-06-29T20:39:09Z
2015-06-29T20:39:09
2016-12-29T17:45:09Z
2016-12-29T17:45:09
```

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

14.3 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"
}
```

14.4 number *type*

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

14.5 string *type*

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

14.6 URL *type*

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

15 Changelog

15.1 Changes between OCPI 2.1 and 2.1.1

Lots of typos fixed and textual improvements.

The following changes to messages/objects etc.

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
CDRs / CDR object	Minor / Minor	Minimal / Minimal	field: CDR.id is changed from string(15) to string(36).
CDRs / CDR object	Minor / Minor	Minimal / Minimal	field: CDR.auth_id is changed from string(32) to string(36).
CDRs / CDR object	Minor / Minor	Minimal / Minimal	field: Session.stop_date_time is changed from optional (?) to required (1).
Commands / ReserveNow object	Minor / Minor	Minimal / Minimal	field: ReserveNow.location_id is changed from string(15) to string(39).
Commands / ReserveNow object	Minor / Minor	Minimal / Minimal	field: ReserveNow.evse_uid is changed from string(15) to string(39).
Commands / StartSession object	Minor / Minor	Minimal / Minimal	field: StartSession.location_id is changed from string(15) to string(39).
Commands / StartSession object	Minor / Minor	Minimal / Minimal	field: StartSession.evse_uid is changed from string(15) to string(39).
Commands / StopSession object	Minor / Minor	Minimal / Minimal	field: StopSession.session_id is changed from string(15) to string(36).
Commands / UnlockConnector object	Minor / Minor	Minimal / Minimal	field: UnlockConnector.location_id is changed from string(15) to string(39).
Commands / UnlockConnector object	Minor / Minor	Minimal / Minimal	field: UnlockConnector.evse_uid is changed from string(15) to string(39).
Commands / UnlockConnector object	Minor / Minor	Minimal / Minimal	field: UnlockConnector.connector_id is changed from string(15) to string(36).
Locations / CPO GET Object method	Minor / Minor	Minimal / Minimal	parameter: location_id is changed from string(15) to string(39).
Locations / CPO GET Object method	Minor / Minor	Minimal / Minimal	parameter: evse_uid is changed from string(15) to string(39).
Locations / CPO GET Object method	Minor / Minor	Minimal / Minimal	parameter: connector_id is changed from string(15) to string(36).
Locations / eMSP GET method	Minor / Minor	Minimal / Minimal	parameter: location_id is changed from string(15) to string(39).
Locations / eMSP GET method	Minor / Minor	Minimal / Minimal	parameter: evse_uid is changed from string(15) to string(39).
Locations / eMSP GET method	Minor / Minor	Minimal / Minimal	parameter: connector_id is changed from string(15) to string(36).
Locations / eMSP PUT method	Minor / Minor	Minimal / Minimal	parameter: location_id is changed from string(15) to string(39).
Locations / eMSP PUT method	Minor / Minor	Minimal / Minimal	parameter: evse_uid is changed from string(15) to string(39).
Locations / eMSP PUT method	Minor / Minor	Minimal / Minimal	parameter: connector_id is changed from string(15) to string(36).
Locations / Location object	Minor / Minor	Minimal / Minimal	field: Location.id is changed from string(15) to string(39).
Locations / EVSE object	Minor / Minor	Minimal / Minimal	field: EVSE.uid is changed from string(15) to string(39).
Locations / Connector object	Minor / Minor	Minimal / Minimal	field: Connector.id is changed from string(15) to string(36).
Sessions / eMSP GET method	Minor / Minor	Minimal / Minimal	parameter: session_id is changed from string(15) to string(36).
Sessions / eMSP PUT method	Minor / Minor	Minimal / Minimal	parameter: session_id is changed from string(15) to string(36).

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Sessions / Session object	Minor / Minor	Minimal / Minimal	field: Session.id is changed from string(15) to string(36).
Sessions / Session object	Minor / Minor	Minimal / Minimal	field: Session.auth_id length changed from 15 to 36 this was THE bug in 2.1.
Sessions / Session object	Minor / Minor	Minimal / Minimal	field: Session.total_cost is changed from required (1) to optional (?).
Tariffs / eMSP GET method	Minor / Minor	Minimal / Minimal	parameter: tariff_id is changed from string(15) to string(36).
Tariffs / eMSP PUT method	Minor / Minor	Minimal / Minimal	parameter: tariff_id is changed from string(15) to string(36).
Tariffs / eMSP DELETE method	Minor / Minor	Minimal / Minimal	parameter: tariff_id is changed from string(15) to string(36).
Tariffs / Tariff object	Minor / Minor	Minimal / Minimal	field: Tariff.id length changed from string(15) to string(36).
Tokens / CPO GET method	Minor / Minor	Minimal / Minimal	parameter: token_uid is changed from string(15) to string(36).
Tokens / CPO PUT method	Minor / Minor	Minimal / Minimal	parameter: token_uid is changed from string(15) to string(36).
Tokens / eMSP POST method	Minor / Minor	Minimal / Minimal	parameter: token_uid is changed from string(15) to string(36).
Tokens / Token object	Minor / Minor	Minimal / Minimal	field: Token.uid length changed from string(15) to string(36).
Tokens / Token object	Minor / Minor	Minimal / Minimal	field: Token.auth_id length changed from string(32) to string(36).
Transport and Format / Response format	Minor / Minor	Minimal / Minimal	field: data now allows String as possible type, needed for the commands module.

15.2 Changes between OCPI 2.0 and 2.1

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
CDRs / CDR object	Major / Major	Minimal / Minimal	replaced field: “total_usage” with: “total_energy”, “total_time” and “total_parking_time”
CDRs / CDR object	Major / Major	Minimal / Minimal	OCPI decimal type is removed and replaced by JSON number.
CDRs / CDR object	Major / Major	Average / Average	new field added: “last_updated”, GET method filters changed to use this new field instead of start of charging session.
CDRs / CdrDimension class	Major / Major	Minimal / Minimal	OCPI decimal type is removed and replaced by JSON number.
CDRs / CdrDimension class	Minor / Minor	Minimal / Minimal	Generic DimensionType replaced by CdrDimensionType.
Credentials / Credentials object	Minor / Minor	Minimal / Minimal	field: “Token” had no max string length, is now set to 64.
Commands module	Optional / Optional	Large / Large	added new commands module.
Locations / Location object	Average / Optional	Minimal / Minimal	new field added: “owner”
Locations / Location object	Average / Optional	Minimal / Minimal	new field added: “time_zone”
Locations / Location object	Minor / Optional	Average / Average	new field added: “facilities”, a list of new type: Facility
Locations / Location object	Minor / Optional	Average / Average	new field added: “energy_mix”
Locations / Location object	Minor / Minor	Minimal / Minimal	new field added: “last_updated”

Context (Module / Object)	Expected Impact: eMSP / CPO	Expected Effort: eMSP / CPO	Description
Locations / EVSE object	Minor / Minor	Minimal / Minimal	new field added: “last_updated”
Locations / Connector object	Minor / Minor	Minimal / Minimal	new field added: “last_updated”
Locations / Connector object	Minor / Minor	Minimal / Minimal	removed field: “status”
Locations / GET list method	Optional / Average	Minimal / Average	added filters to retrieve only Locations that have been updated between date_to/date_from.
Locations / GET object method	Optional / Average	Average / Average	added functionality to retrieve information about a specific Location, EVSE or Connector from a CPO. This can be useful for eMSPs that require ‘real-time’ authorization of Tokens.
Locations / Capability enum	Minor / Optional	Minimal / Minimal	added new values to the enum: Capability.
Sessions / Session object	Major / Major	Minimal / Minimal	OCPI decimal type is removed and replaced by JSON number.
Sessions / Session object	Major / Major	Average / Average	new field added: “last_updated”, GET method filters changed to use this new field instead of start of charging session.
Sessions / eMSP DELETE method	Minor / Optional	Minimal / Minimal	Session DELETE method is removed.
Tariffs / Tariff object	Minor / Optional	Average / Average	new field added: “energy_mix”
Tariffs / Tariff object	Minor / Minor	Minimal / Minimal	new field added: “last_updated”
Tariffs / PriceComponent class	Major / Major	Minimal / Minimal	OCPI decimal type is removed and replaced by JSON number.
Tariffs / PriceComponent class	Major / Major	Minimal / Minimal	OCPI decimal type is removed and replaced by JSON number.
Tariffs / PriceComponent class	Minor / Minor	Minimal / Minimal	Generic DimensionType replaced by TariffDimensionType.
Tariffs / CPO GET method	Optional / Average	Minimal / Average	added filters to retrieve only Tokens that have been updated between date_to/date_from.
Tokens / eMSP POST method	Optional / Major	Large / Large	added functionality for ‘real-time’ authorization of Tokens.
Tokens / Token object	Optional / Minor	Minimal / Average	new field added: language.
Tokens / Token object	Major / Major	Minimal / Average	changed field: whitelist_allowed (type: boolean) to whitelist (type: WhitelistType)
Tokens / Token object	Minor / Minor	Minimal / Minimal	new field added: “last_updated”
Tokens / Token object	Optional / Minor	Minimal / Minimal	field: “visual_number” is now optional.
Tokens / eMSP GET method	Average/ Optional	Average / Minimal	added filters to retrieve only Tokens that have been updated between date_to/date_from.
Version information / Custom Modules	Optional / Optional	Average / Average	added description on how to add custom/customized modules to OCPI.
Version information / Version class	Minor / Minor	Minimal / Minimal	OCPI Version changed from OCPI decimal to VersionNumber enum.
Version information / Version details endpoint	Minor / Minor	Minimal / Minimal	OCPI Version changed from OCPI decimal to VersionNumber enum.