

OCPI 2.0

Open Charge Point Interface 2.0

<https://github.com/ocpi>

24.12.2015

Contents

1	OCPI	6
1.1	Introduction and background	6
2	Terminology and Definitions	7
2.1	Abbreviations	7
2.2	Provider and Operator abbreviation	7
2.2.1	The Netherlands	7
2.2.2	Germany	7
2.2.3	Austria	7
2.3	Charging topology	7
2.4	Variable names	8
2.5	Cardinality	8
3	Transport and format	9
3.1	JSON / HTTP implementation guide	9
3.1.1	Security and authentication	9
3.1.2	Request format	9
3.1.3	Client owned object push	10
3.1.4	Response format	11
3.2	Interface endpoints	12
4	Status codes	13
4.1	1xxx: Success	13
4.2	2xxx: Client errors	13
4.3	3xxx: Server errors	13
5	Version information endpoint	15
5.1	Data	15
5.1.1	Version <i>class</i>	15
5.2	GET	15
5.2.1	Example	15
6	Version details endpoint	16
6.1	Data	16
6.1.1	Endpoint <i>class</i>	16
6.1.2	ModuleID <i>enum</i>	16
6.2	GET	16
6.2.1	Example	16

7	Credentials endpoint	17
7.1	Data	17
7.1.1	Credentials object	17
7.1.2	Example	17
7.2	GET	18
7.3	POST	18
7.4	PUT	18
7.5	DELETE	18
7.6	Use cases	18
7.6.1	Registration	18
7.6.2	Updating to a newer version	20
7.6.3	Changing endpoints for the current version	20
7.6.4	Errors during handshake	20
7.6.5	Required endpoints not available	21
8	<i>Locations</i> module	22
8.1	Flow and Lifecycle	22
8.2	Interfaces and endpoints	22
8.2.1	CPO Interface	22
8.2.2	eMSP Interface	23
8.3	Object description	26
8.3.1	<i>Location</i> Object	26
8.3.2	<i>EVSE</i> Object	28
8.3.3	<i>Connector</i> Object	29
8.4	Data types	30
8.4.1	<i>AdditionalGeoLocation</i> <i>class</i>	30
8.4.2	<i>BusinessDetails</i> <i>class</i>	30
8.4.3	<i>Capability</i> <i>enum</i>	30
8.4.4	<i>ConnectorFormat</i> <i>enum</i>	31
8.4.5	<i>ConnectorType</i> <i>enum</i>	31
8.4.6	<i>ExceptionalPeriod</i> <i>class</i>	31
8.4.7	<i>GeoLocation</i> <i>class</i>	32
8.4.8	<i>Hours</i> <i>class</i>	32
8.4.9	<i>Image</i> <i>class</i>	32
8.4.10	<i>ImageCategory</i> <i>enum</i>	33
8.4.11	<i>LocationType</i> <i>enum</i>	33
8.4.12	<i>ParkingRestriction</i> <i>enum</i>	34
8.4.13	<i>PowerType</i> <i>enum</i>	34
8.4.14	<i>RegularHours</i> <i>class</i>	34
8.4.15	<i>Status</i> <i>enum</i>	35
8.4.16	<i>StatusSchedule</i> <i>class</i>	36

9	<i>Sessions</i> module	37
9.1	Flow and Lifecycle	37
9.1.1	Push model	37
9.1.2	Pull model	37
9.2	Interfaces and endpoints	37
9.2.1	CPO Interface	37
9.2.2	eMSP Interface	38
9.3	Object description	40
9.3.1	<i>Session</i> Object	40
9.4	Data types	42
9.4.1	SessionStatus <i>enum</i>	42
10	<i>CDRs</i> module	43
10.1	Flow and Lifecycle	43
10.1.1	Push model	43
10.1.2	Pull model	43
10.2	Interfaces and endpoints	43
10.2.1	CPO Interface	43
10.2.2	eMSP Interface	44
10.3	Object description	45
10.3.1	<i>CDR</i> Object	45
10.4	Data types	47
10.4.1	AuthMethod <i>enum</i>	47
10.4.2	CdrDimension <i>class</i>	47
10.4.3	ChargingPeriod <i>class</i>	47
11	<i>Tariffs</i> module	49
11.1	Flow and Lifecycle	49
11.1.1	Push model	49
11.1.2	Pull model	49
11.2	Interfaces and endpoints	49
11.2.1	CPO Interface	49
11.2.2	eMSP Interface	50
11.3	Object description	52
11.3.1	<i>Tariff</i> Object	52
11.4	Data types	55
11.4.1	DayOfWeek <i>enum</i>	55
11.4.2	PriceComponent <i>class</i>	55
11.4.3	TariffElement <i>class</i>	55
11.4.4	TariffRestrictions <i>class</i>	55

12	<i>Tokens</i> module	56
12.1	Flow and Lifecycle	56
12.1.1	Push model	56
12.1.2	Pull model	56
12.2	Interfaces and endpoints	57
12.2.1	CPO Interface	57
12.2.2	eMSP Interface	58
12.3	Object description	59
12.3.1	<i>Token</i> Object	59
12.4	Data types	60
12.4.1	TokenType <i>enum</i>	60
12.5	Types	60
12.5.1	CiString <i>type</i>	60
12.5.2	DateTime <i>type</i>	60
12.5.3	decimal <i>type</i>	60
12.5.4	DimensionType <i>enum</i>	60
12.5.5	DisplayText <i>class</i>	61
12.5.6	string <i>type</i>	61
12.5.7	URL <i>type</i>	61

1 OCPI

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

An international group of companies already supports OCPI. Initiators are EV Box, The New Motion, ElaadNL, BeCharged, Greenflux and Last Mile Solutions. Other participants include Next Charge, Freshmile, Plugsurfing, Charge-partner, Hubeject, eClearing.net 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 this model, the provider and the operator play an important role. 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>

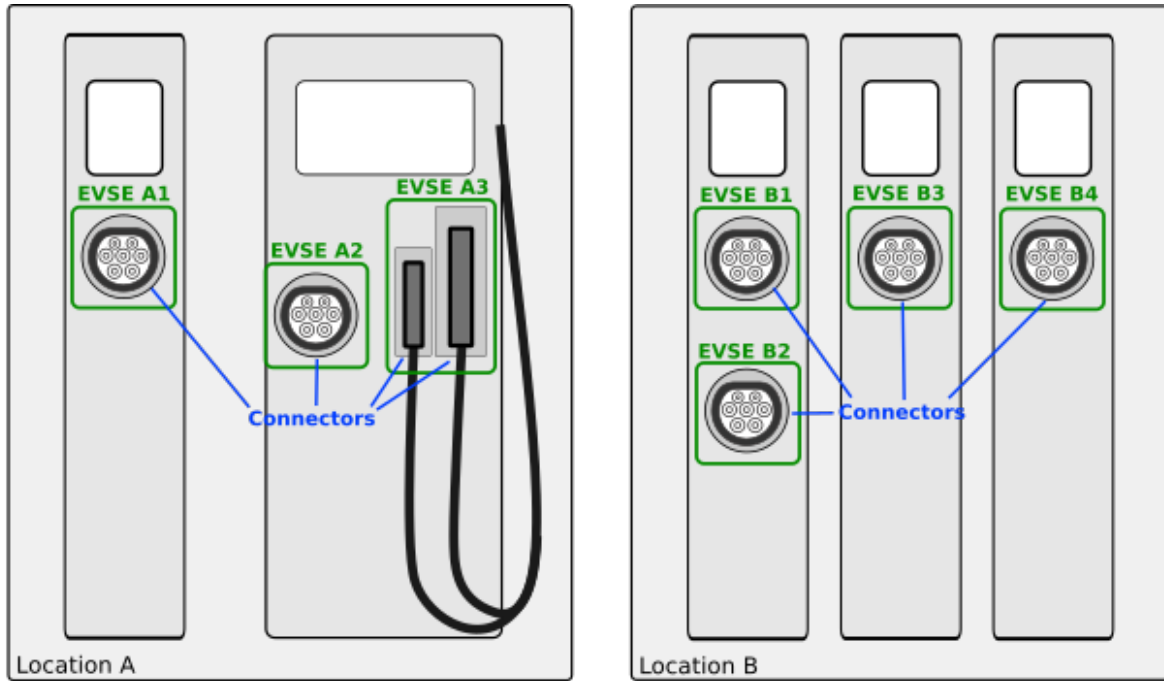
2.2.3 Austria

Austrian Mobile Power GmbH maintains a registry for Austria. The list is not public available. For more information visit [austrian-mobile-power.at](#)

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.	Object
1	Required object.	Object
*	A list of zero or more objects.	[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 Request format

Each HTTP request must add a 'Authorization' header. The header looks as following:

```
Authorization: Token IpbJ0Xxkx0AuKR92z0nEcmVF3Qw09VG7I7d/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. The server can then 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 similar way as REST APIs do.

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.

3.1.2.1 GET

All GET methods that return a list of objects have pagination.

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

3.1.2.1.1 Paginated Request

The following table is a list of 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.
limit	Maximum number of objects to GET. Note: the server might decided to return less objects, because there are no more objects or the server limits the maximum amount of object to return to prevent, for example, overloading the system

3.1.2.1.2 Paginated Response

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

HTTP Parameter	Description
link	Link to the 'next' page should be provided, see example below
X-Total-Count	(Custom HTTP Header) Total number of objects available in the server 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, then less than this number will be returned.

Example of a required OCPI pagination link header

Link: <<https://www.server.com/ocpi/cpo/2.0/cdrs/?offset=5&limit=50>>; rel="next"

3.1.2.2 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.

3.1.2.3 PATCH

A PATCH request must only specify the object's identifier 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 fails, the client is expected to call the GET method to check the state of the object in the other parties system. If the object doesn't exist, the client should do a **PUT**.

3.1.3 Client owned object push

Normal client/server RESTful services work in a way that the Server is the owner of objects that are created. The client request 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 only request 1 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 1 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.

POST is not supported for these kind of modules.

PUT is used to send new objects to the servers.

The distinction between objects from different CPOs/eMSPs is made, based on a `{country-code}` and `{party-id}`.

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>

3.1.4 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 a 'application/json' type and contains a JSON object with the following properties:

Property	Type	Card.	Description
data	object	*	Contains the actual response data object or list of objects from each request.
status_code	Integer	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 example 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.

3.1.4.1 Example: Version details response

```
{
  "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"
}
```

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

```
{
  "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"
}
```

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

```
{
  "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"
}
```

3.2 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

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.

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

Code	Description
2000	Generic client error
2001	Invalid or missing parameters

4.3 3xxx: Server errors

Error during processing of the OCPI payload in the server. The message was syntactically right 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 handshake: When the initializing party requests data from the other party during the open POST call to it's credentials endpoint. If one of the GETs can not be processed, the party should return this error in the POST response.

Code	Description
3002	Unsupported version.
3003	No matching endpoints or expected endpoints missing between parties. Used during the handshake 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	decimal	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": "1.9",
    "url": "https://example.com/ocpi/cpo/1.9/"
  },
  {
    "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	decimal	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.

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

Identifier: `credentials`

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

Method	Description
GET	Retrieves the client's credentials for the server's platform.
POST	Provides the server with credentials to the client's system (i.e. register).
PUT	Updates the server's credentials to the client's system.
DELETE	Informs the server that its credentials to the client's system are now invalid (i.e. unregister).

7.1 Data

7.1.1 Credentials object

Property	Type	Card.	Description
token	String	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	<code>string(3)</code>	1	CPO or eMSP ID of this party. (following the 15118 ISO standard).
country_code	<code>string(2)</code>	1	Country code of the country this party is operating in.

7.1.2 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"
  }
}
```

7.2 GET

Retrieves the client's credentials for the server's platform.

7.3 POST

Provides the server with credentials to the client's system, this initiates the registration process. If successful, the server responds with the client's new credentials to the server's system.

7.4 PUT

Updates the server's credentials to the client's system and switch to the version that contains this credentials endpoint. The server should also fetch the client's endpoints for this version.

If successful, the server responds with the client's (potentially updated) credentials to the server's system.

7.5 DELETE

Informs the server that its credentials to the client's system are now invalid and can no longer be used. This is the unregistration process.

If successful, the server responds with an empty OCPI response message (i.e. `data` is null).

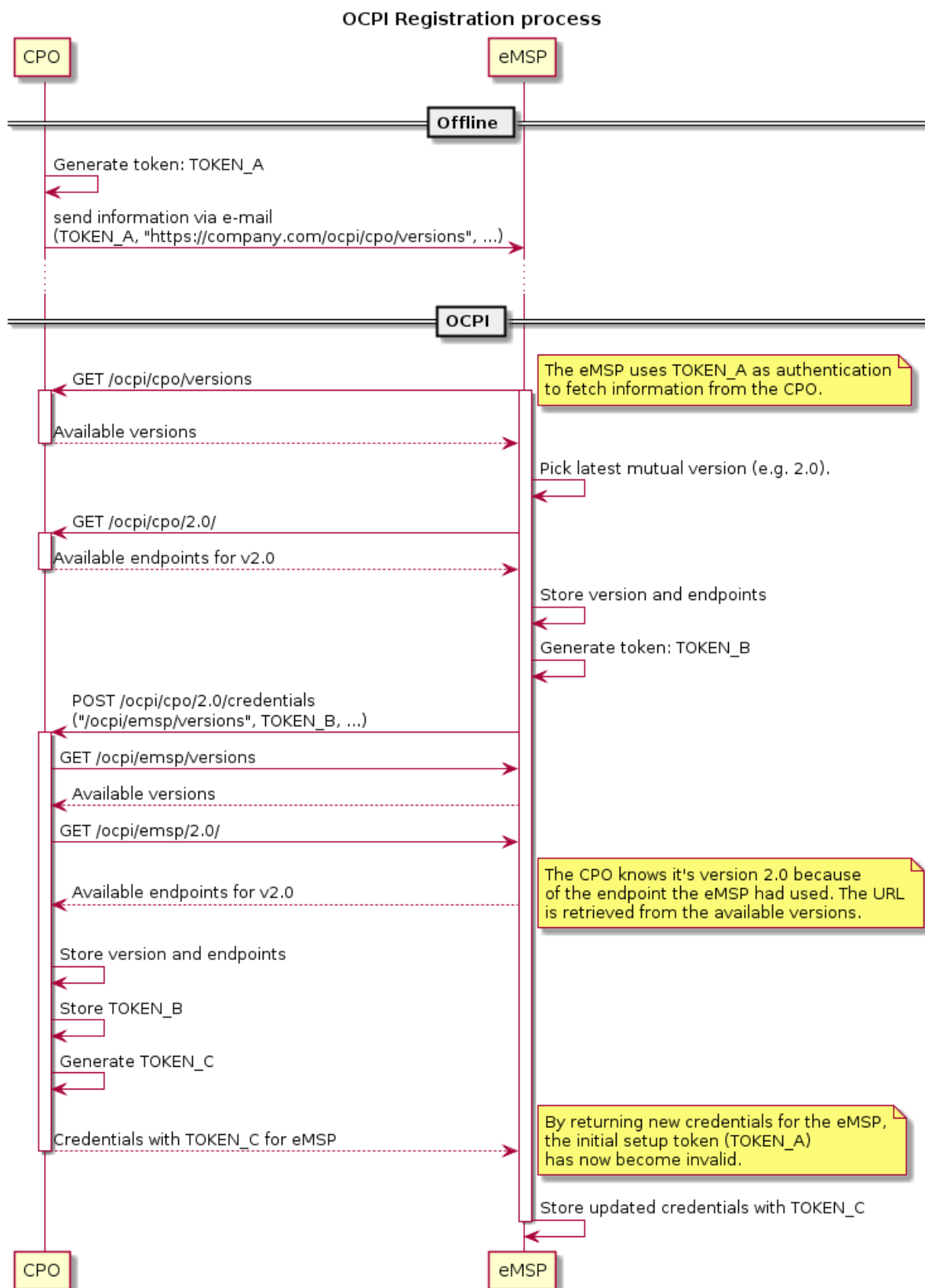
7.6 Use cases

7.6.1 Registration

To register a CPO in a 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 will have to be sent to the eMSP in some 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.

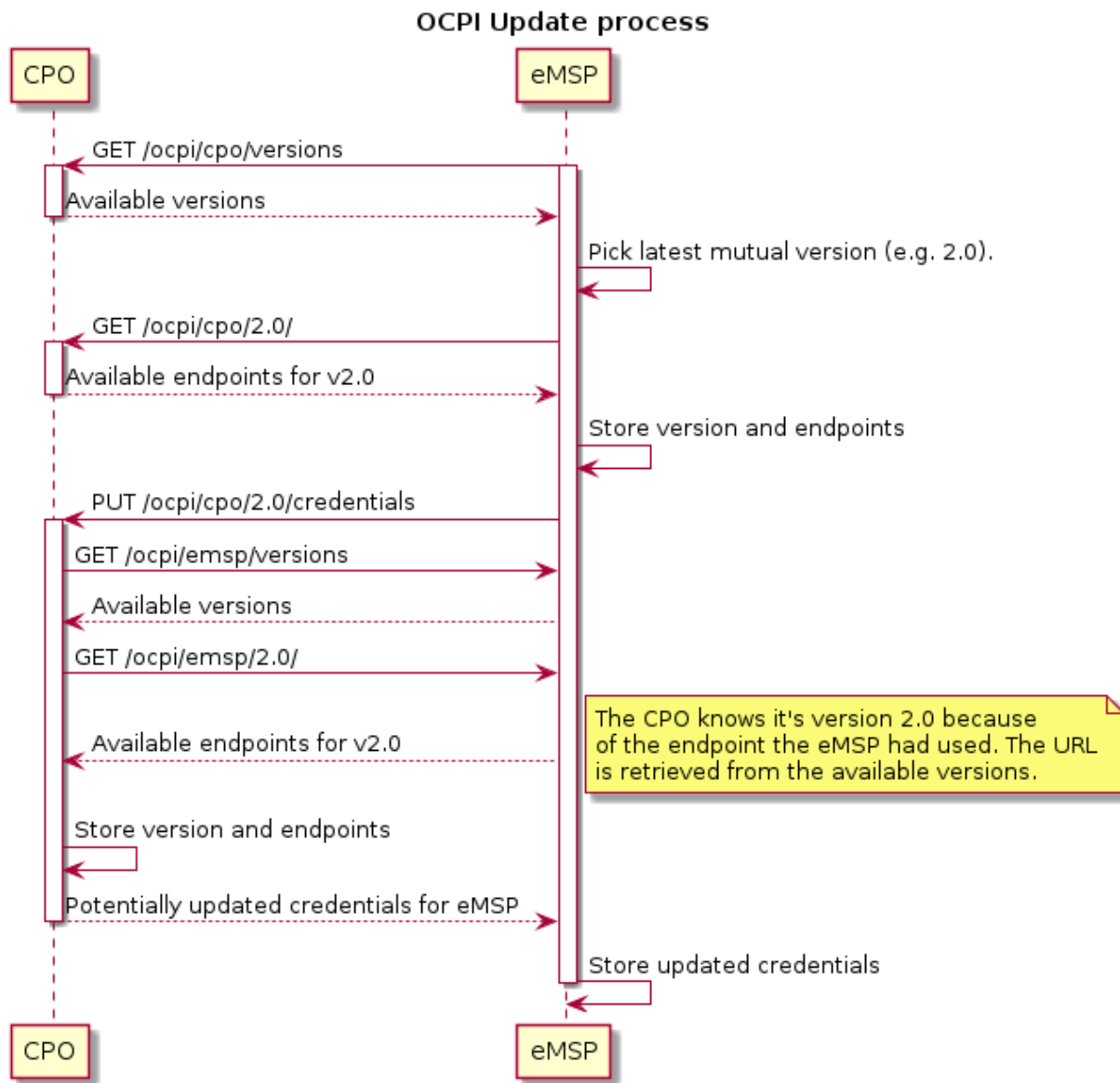
(In the sequence diagrams below we use relative paths as short resource identifiers to illustrate a point; please note that they should really be absolute URLs in any working implementation of OCPI)



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

7.6.2 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.



7.6.3 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.

7.6.4 Errors during handshake

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

7.6.5 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 a eMSP system.

In case the client starting the credentials exchange process, cannot find the endpoints it expects. It is expected to NOT 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. 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.

8.1 Flow and Lifecycle

The Location 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 they push 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 a Location, EVSE or Connector, they must update by setting the `status` field to `REMOVED` and call the [PUT](#) or [PATCH](#) on the eMSP system.

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 correct any more.

8.2.1 CPO Interface

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

Method	Description
GET	Fetch all available locations. (paginated)
POST	n/a
PUT	n/a
PATCH	n/a
DELETE	n/a

8.2.1.1 GET Method

Fetch information about all available locations and EVSEs at this CPO.

8.2.1.1.1 Request Parameters

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

Parameter	Datatype	Required	Description
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 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 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
Location	*	List of all locations with valid EVSEs.

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 of was missing an object.

8.2.2.1.1 Request Parameters

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	int	yes	Location.id of the Location object to retrieve.

Parameter	Datatype	Required	Description
evse_uid	int	no	Evse.uid, required when requesting an Evse or Connector object.
connector_id	int	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
<i>Choice: one of three</i>		
> 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 a eMSP from different CPOs, the **{party_id}** and **{country_code}** have to be included in the URL.

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	int	yes	Location.id of the new Location object, or the Location of which an EVSE or Location object is send
evse_uid	int	no	Evse.uid, required when an Evse or Connector object is send/replaced.
connector_id	int	no	Connector.id, required when a Connector object is send/replaced.

8.2.2.3 Request Body

Type	Card.	Description
<i>Choice: one of three</i>		
> 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.

8.2.2.4 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.

8.2.2.4.1 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",
}
```

8.2.2.4.2 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",
}
```

8.2.2.4.3 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"
}
```

8.2.2.4.4 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,
}
```

8.2.2.4.5 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>

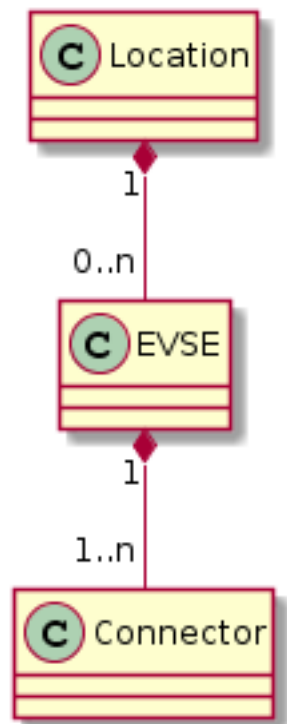
```
{  
  "status": "REMOVED",  
}
```

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 to following relation.

Locations class diagram



8.3.1 Location Object

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

A *Location* without valid *EVSE* objects can be considered as expired and should no longer be displayed.

Property	Type	Card.	Description
id	string(15)	1	Uniquely identifies the location within the CPOs platform (and suboperator platforms).

Property	Type	Card.	Description
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.
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.

8.3.1.1 Example

```
{
  "id": "LOC1",
  "type": "on_street",
  "name": "Gent Zuid",
  "address": "F.Roosevelttlaan 3A",
  "city": "Gent",
  "postal_code": "9000",
  "country": "BEL",
  "coordinates": {
    "latitude": "3.72994",
    "longitude": "51.04759"
  },
  "evses": [{
    "uid": "3256",
    "id": "BE-BEC-E041503001",
    "status": "AVAILABLE",
    "capabilities": [
      "RESERVABLE"
    ]
  }],
}
```

```

    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "CABLE",
      "power_type": "AC_3_PHASE",
      "voltage": 220,
      "amperage": 16,
      "tariff_id": "11"
    }, {
      "id": "2",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "voltage": 220,
      "amperage": 16,
      "tariff_id": "11"
    }],
    "physical_reference": 1,
    "floor_level": "-1"
  }, {
    "uid": "3257",
    "id": "BE-BEC-E041503002",
    "status": "reserved",
    "capabilities": [
      "RESERVABLE"
    ],
    "connectors": [{
      "id": "1",
      "standard": "IEC_62196_T2",
      "format": "SOCKET",
      "power_type": "AC_3_PHASE",
      "voltage": 220,
      "amperage": 16,
      "tariff_id": "12"
    }],
    "physical_reference": 2,
    "floor_level": -2
  }],
  "operator": {
    "name": "BeCharged"
  }
}

```

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 may be used at a time. The list of connectors is seen as atomic. This implies that for any changes or updates to a single connector, the full list of all connectors will have to be specified. Any connector not on that list is considered as deleted.

Property	Type	Card.	Description
uid	<code>string</code> (15)	1	Uniquely identifies the EVSE within the CPOs platform (and suboperator platforms). For example a database unique ID
evse_id	<code>string</code> (48)	?	ISO 15118 Compliant EVSE ID. 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	<code>Status</code>	1	Indicates the current status of the EVSE.
status_schedule	<code>StatusSchedule</code>	*	Indicates a planned status in the future of the EVSE.
capabilities	<code>Capability</code>	*	List of functionalities that the EVSE is capable of.
connectors	<code>Connector</code>	+	List of available connectors on the EVSE.
floor_level	<code>string</code> (4)	?	Level on which the charging station is located (in garage buildings) in the locally displayed numbering scheme.
coordinates	<code>GeoLocation</code>	?	Coordinates of the EVSE.
physical_reference	<code>string</code> (16)	?	A number/string printed on the outside of the EVSE for visual identification.
directions	<code>DisplayText</code>	*	Multi-language human-readable directions when more detailed information on how to reach the EVSE from the <i>Location</i> is required.
parking_restrictions	<code>ParkingRestriction</code>	*	The restrictions that apply to the parking spot.
images	<code>Image</code>	*	Links to images related to the EVSE such as photos or logos.

8.3.3 Connector Object

A connector is the socket or cable available for the EV to make use of. 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	<code>string</code> (15)	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 <i>EVSE</i> object.
status	<code>Status</code>	1	Indicates the current status of this connector. standard <code>ConnectorType</code> 1
format	<code>ConnectorFormat</code>	1	The standard of the installed connector. The format (socket/cable) of the installed connector.
power_type	<code>PowerType</code>	1	
voltage	<code>int</code>	1	Voltage of the connector (line to neutral for AC_3_PHASE), in volt [V].

Property	Type	Card.	Description
amperage	int	1	maximum amperage of the connector, in ampere [A].
tariff_id	string(15)	?	Identifier of the current charging tariff structure
terms_and_conditionsURL		?	URL to the operator's terms and conditions.

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: <code>-?[0-9]\{1,2\}\.[0-9]\{6\}</code>
longitude	string(11)	1	Longitude of the point in decimal degree. Example: -126.104965. Decimal separator: “.” Regex: <code>-?[0-9]\{1,3\}\.[0-9]\{6\}</code>
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
RESERVABLE	The EVSE can be reserved.
RFID_READER	Charging at this EVSE can be authorized with a RFID token

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 a 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 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.

Field Name	Field Type	Card.	Description
------------	------------	-------	-------------

8.4.7 GeoLocation *class*

Property	Type	Card.	Description
latitude	string (10)	1	Latitude of the point in decimal degree. Example: 50.770774. Decimal separator: “.” Regex: <code>-?[0-9]\{1,2\}\.[0-9]\{6\}</code>
longitude	string (11)	1	Longitude of the point in decimal degree. Example: -126.104965. Decimal separator: “.” Regex: <code>-?[0-9]\{1,3\}\.[0-9]\{6\}</code>

8.4.8 Hours *class*

Opening and access hours for the location.

Field Name	Field Type	Card.	Description
<i>Choice: one of two</i>			
> 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 per day and 7 days per 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.

8.4.9 Image *class*

This class references images related to a EVSE in terms of a file name or uri. 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 a 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 are minimum 800 pixels wide and 600 pixels height. Thumbnail representations for photos should always have the same orientation than 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 than 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 web browser.
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

8.4.10 ImageCategory enum

The category of an image to obtain the correct usage in an user presentation. Has to be set accordingly to the image content in order to guaranty 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 a 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 municipal, 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

8.4.11 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).

8.4.12 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 allowed only 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.

8.4.13 PowerType *enum*

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

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

8.4.14 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.

8.4.14.1 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	-	-

8.4.15 Status *enum*

The status of an EVSE.

Value	Description
AVAILABLE	The EVSE is able to start a new charging session.
BLOCKED	The EVSE not accessible because of a physical barrier, i.e. a car.
CHARGING	The EVSE is in use.
INOPERATIVE	The EVSE is not yet active or it is no longer available (deleted).
OUTOFORDER	The EVSE is currently out of order.
PLANNED	The EVSE is planned, will be operating soon
REMOVED	The EVSE/charge point is discontinued/removed.
RESERVED	The EVSE is reserved for a particular EV driver and is unavailable for other drivers.
UNKNOWN	No status information available. (Also used when offline)

8.4.16 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 from 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 push 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 send to the eMSP system by calling **PATCH** on the eMSPs Sessions endpoint with the updated Session object

When the CPO deletes a Session, they will update the eMSPs systems by calling **DELETE** on the eMSPs Sessions endpoint, on the Sessions unique URL.

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.

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 object of charging sessions started 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. Only Sessions with a start date/time between the given {date_from} and {date_to} will be returned.

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

Parameter	Required	Description
Datatype		
date_from DateTime	yes	Begin charging session start Date/Time of the Sessions to fetch.

Parameter	Datatype	Required	Description
date_to	DateTime	no	End charging session start Date/Time of the Sessions to fetch, if omitted all Sessions up to now are request to be returned.
offset	int	no	The offset of the first object returned. Default is 0.
limit	int	no	Maximum number of objects to GET.

NOTE: The CPO is allowed to return a (not specified) maximum amount of Sessions, to prevent overloading there system. In this version of OCPI it is not possible to detect if the CPO returned not all Sessions that match the filter.

9.2.1.1.1 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}/{tariff_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	Delete the Session object of id {session-id}.

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

Parameter	Datatype	Required	Description
country_code	string (2)	yes	Country code of the CPO requesting this PUT to the eMSP system.

Parameter	Datatype	Required	Description
party_id	string(3)	yes	Party ID (Provider ID) of the CPO requesting this PUT to the eMSP system.
session-id	string(15)	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

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(15)	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"
}
```

9.2.2.4 DELETE Method

Inform the eMSP about a deleted Session object.

9.2.2.4.1 Request Parameters

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(15)	yes	ID of the Session to be deleted

9.3 Object description

9.3.1 Session Object

Property	Type	Card.	Description
id	string(15)	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	decimal	1	How many kWh are charged.
auth_id	string(15)	1	An id provided by the authentication used, so that the eMSP knows to which driver the session belongs.
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	decimal	1	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.
status	SessionStatus	1	The status of the session.

9.3.1.1 Examples

9.3.1.2 Simple Session example of a just starting session

```
{
  "id": "101",
  "start_datetime": "2015-06-29T22:39:09+02:00",
  "kwh": "0.00",
```



```

"auth_id": "FA54320",
"location": {
  "id": "LOC1",
  "type": "on_street",
  "name": "Gent Zuid",
  "address": "F.Roosevelttlaan 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"
    }]
  },
},
"currency": "EUR",
"total_cost": "2.50",
"status": "PENDING"
}

```

9.3.1.2.1 Simple Session example of a short finished session

```

{
  "id": "101",
  "start_datetime": "2015-06-29T22:39:09+02:00",
  "end_datetime": "2015-06-29T23:50:16+02:00",
  "kwh": "0.00",
  "auth_id": "FA54320",
  "location": {
    "id": "LOC1",
    "type": "on_street",
    "name": "Gent Zuid",
    "address": "F.Roosevelttlaan 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"
        }]
    },
    "connector_id": "1",
    "currency": "EUR",
    "charging_periods": [{
        "start_date_time": "2015-06-29T22:39:09+02:00",
        "dimensions": [{
            "type": "ENERGY",
            "volume": 120
        }, {
            "type": "MAX_CURRENT",
            "volume": 30
        }]
    }, {
        "start_date_time": "2015-06-29T22:40:54+02:00",
        "dimensions": [{
            "type": "energy",
            "volume": 41000
        }, {
            "type": "MIN_CURRENT",
            "volume": 34
        }]
    }, {
        "start_date_time": "2015-06-29T23:07:09+02:00",
        "dimensions": [{
            "type": "PARKING_TIME",
            "volume": 0.718
        }]
    }],
    "total_cost": "8.50",
    "status": "COMPLETED"
}

```

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 has 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 then for example once a month, that is also allowed by OCPI.

10.1 Flow and Lifecycle

CDRs are created by the CPO. They are probably only send 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 send 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 create CDRs(s)

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 might calculate its own costs for billing their customer. An eMSP might have a very different contract/pricing model with the EV driver then the tariff structure from the CPO.

NOTE: CDRs cannot 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.

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 of charging sessions started between the {date_from} and {date_to} (paginated)
POST	n/a
PUT	n/a

Method	Description
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 of charging sessions with a start date/time 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	Begin charging session start Date/Time of CDRs to fetch.
date_to	DateTime	no	End charging session start Date/Time of CDRs to fetch, if omitted all CDRs up to now are request to be returned.
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 send.

Type	Card.	Description
CDR	1	New CDR object.

10.2.2.2.2 Response Headers

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

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(15)	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	?	Stop timestamp of the charging session.
auth_id	string(32)	1	Reference to a token, identified by the auth_id field of the Token.
auth_method	AuthMethod	1	Method used for authentication.

Property	Type	Card.	Description
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	Tariff	*	List of relevant tariff elements, see: Tariffs .
charging_periods	ChargingPeriod	+	List of charging periods that make up this charging session. A session consists of 1 or more periods with, each period has a different relevant Tariff.
total_cost	decimal	1	Total cost of this transaction.
total_usage	CdrDimension	*	List of total usage elements, for example: total parking time and total energy charged.
remark	string (255)	?	Optional remark, can be used to provide additional human readable information to the CDR, for example: reason why a transaction was stopped.

10.3.1.1 Example of a CDR

```
{
  "id": "12345",
  "start_date_time": "2015-06-29T21:39:09+02:00",
  "stop_date_time": "2015-06-29T23:37:32+02:00",
  "auth_id": "FA54320",
  "auth_method": "WHITELIST",
  "location": {
    "id": "LOC1",
    "type": "on_street",
    "name": "Gent Zuid",
    "address": "F.Roosevelttlaan 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"
    }]
  },
  "currency": "EUR",
  "tariffs": [{
```

```

    "id": "12",
    "currency": "EUR",
    "elements": [{
      "price_components": [{
        "type": "TIME",
        "price": "2.00",
        "step_size": 300
      }]
    }],
    "charging_periods": [{
      "start_date_time": "2015-06-29T21:39:09+02:00",
      "dimensions": [{
        "type": "TIME",
        "volume": 1.973
      }]
    }],
    "total_cost": "4,00",
    "total_usage": [{
      "type": "TIME",
      "volume": "1.973"
    }, {
      "type": "ENERGY",
      "volume": 15.342
    }]
  }
}

```

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	DimensionType	1	Type of cdr dimension
volume	decimal	1	Volume of the dimension consumed, measured according to the dimension type.

10.4.3 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 over 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 create 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 mend 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 correct anymore.

11.2.1 CPO Interface

The CPO Tariffs interface give the eMSP the ability to request all tariffs.

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

Method	Description
GET	Returns all Tariff Objects from the CPO (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

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

Parameter	Datatype	Required	Description
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 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 Location 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 of was missing an object.

11.2.2.1.1 Request Parameters

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 (15)	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 send.

Type	Card.	Description
Tariff	1	New or updated Tariff object

11.2.2.2.2 Request Parameters

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(15)	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

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.

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

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(15)	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 than 1 element, then 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 none of the TariffElements before this matches the current charging period.

Property	Type	Card.	Description
id	string(15)	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

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
        }]
    }]
}

```

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
    }]
  }]
}

```

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
    }]
  }]
}

```

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:

- 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"]
    }
}

```

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	DimensionType	1	Type of tariff dimension, see: Types
price	decimal	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.

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	?	List of tariff restrictions

11.4.4 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

Property	Type	Card.	Description
end_time	string(5)	?	End time of day, for example 19:45, valid until this time of the day
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	decimal	?	Minimum used energy in kWh, for example 20, valid from this amount of energy is used
max_kwh	decimal	?	Maximum used energy in kWh, for example 50, valid until this amount of energy is used
min_power	decimal	?	Minimum power in kW, for example 0, valid from this charging speed
max_power	decimal	?	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.

They then 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 create 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 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 at 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 method is not for operational flow.

12.2 Interfaces and endpoints

There is both a CPO and an eMSP interface for Tokens. Advised is 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 correct any more.

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 of was missing an object.

12.2.1.1.1 Request Parameters

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.
token_uid	string (15)	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 CPO to the eMSP.

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 Tariff object.

12.2.1.2.2 Request Parameters

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.
token_uid	string(15)	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": "FA54320",
  "visual_number": "DF000-2001-8999",
  "issuer": "TheNewMotion",
  "valid": true,
  "allow_whitelist": true
}
```

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 all Tokens, when needed. It is not possible to validate/request a single token, during normal operation the Token cache of the CPO should always be kept up to date via the CPO Interface.

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

Method	Description
GET	Get the list of known Tokens (paginated)

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

12.2.2.1 GET Method

Fetch information about all Tokens known in the eMSP systems.

12.2.2.1.1 Request Parameters

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

Parameter	Datatype	Required	Description
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.

12.3 Object description

12.3.1 Token Object

Property	Type	Card.	Description
uid	string (15)	1	Identification used by CPO system to identify this token, for example RFID hidden ID
type	TokenType	1	Type of the token
auth_id	string (32)	1	Uniquely identifies the EV Driver contract token within the eMSPs platform (and suboperator platforms).
visual_number	string (64)	1	Visual readable number/identification of the Token
issuer	string (64)	1	Issuing company
valid	boolean	1	Is this Token valid
allow_whitelist	boolean	?	It is allowed to validate a charging session with token without requesting and authorization from the eMSP? , default is FALSE. NOTE: For this release this field always needs to be set to TRUE

The combination of *uid* and *type* should be unique for every token.

12.3.1.1 Example

```
{
  "uid": "012345678",
  "type": "RFID",
  "auth_id": "FA54320",
  "visual_number": "DF000-2001-8999",
  "issuer": "TheNewMotion",
  "valid": true,
  "allow_whitelist": true
}
```

12.4 Data types

12.4.1 TokenType *enum*

Value	Description
OTHER	Other type of token
RFID	RFID Token

12.5 Types

12.5.1 CiString *type*

Case Insensitive String. Only printable ASCII allowed.

12.5.2 DateTime *type*

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

Example:

```
2015-06-29T22:39:09+02:00
2015-06-29T20:39:09Z
2015-06-29T20:39:09
```

12.5.3 decimal *type*

Decimals are formatted as strings following JSONs number format. They are explicitly expressed as strings to make it clear that they should be interpreted as exact decimals and not as floating points or doubles.

Example:

```
"0.68"
"3.1415"
```

12.5.4 DimensionType *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
MIN_CURRENT	defined in A (Ampere), Minimum current
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

12.5.5 DisplayText *class*

Property	Type	Card.	Description
language	<code>string(2)</code>	1	Language Code ISO 639-1
text	<code>string(512)</code>	1	Text to be displayed to a end user. No markup, html etc. allowed.

Example:

```
{
  "language": "en",
  "text": "Standard Tariff"
}
```

12.5.6 string *type*

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

12.5.7 URL *type*

An URL a `string(255)` type following the [w3.org spec](https://www.w3.org/spec/).