OPEN
CHARGE
POINT
INTERFACE

OCPI 3.0-2

Functional Use Cases

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

document version review2, 2024-02-22

Table of Contents

EVRoaming Foundation

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

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

Version History

Version	Date	Author	Description
3.0-review2	2024-03-12	Michel Bayings *EVRoaming Foundation*	Updated introductory texts.
3.0-review1	2024-02-22	Christopher Brown *Stations-e* Daniele Orler *Plugsurfing* Gianfranco de Fabritiis *DCS* Kor Meelker *Chargepoint Inc.* Matthieu Loos *FLO EV Charging* Pieter Goetschalckx *Optimile* Reinier Lamers *ihomer* Robert de Leeuw *EVA Global* Rudolph Froger *TandemDrive* Thomas Fousse *Gireve*	Adding Registration, Request Addressing, Party Issued Objects, and Remote Procedure Calls use cases. Numerous changes to the Functional Modules as noted in the introduction text to each Functional Module.
3.0 Draft 0	21-06-2019	Robert de Leeuw *ihomer*	First documentation structure for OCPI 3.0, moved all existing documentation to the new documents

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

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

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

Examples:

- OCPI 2.1.1 is a different protocol version of OCPI than OCPI 2.1.
- OCPI 2.0-d2 is the same protocol version as OCPI 2.0, but a newer documentation revision: same protocol, newer documentation.

Introduction

This document contains the OCPI 3.0-2 functional use cases. Each functional use case describes how a certain function should be performed by an OCPI platform.

General

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

In the Functional Use Cases we will use the following terms in addition to those defined in the Business Use Cases.

Term	Description
Charge Token	An identifier of an authorization given by an MSP to a driver or group of drivers to charge at one or more EVSEs belonging to one or more CPOs with the MSP taking responsibility of compensating the CPO for the resulting charge sessions.
Party Issued Object	A record of information about an entity managed by an OCPI Party, which is replicated into computer systems of other OCPI parties. Only the party managing the represented entity is to make changes to the record, and the replicas are only updated when replicating changes to the original record.
Hosting	An OCPI Platform is said to *host* an OCPI Party when it is able to offer Party Issued Objects or answer remote procedure calls on behalf of that Party, to at least one other OCPI Platform connected to this first Platform.
Serving	An OCPI Platform is said to *serve* one OCPI Party X to another OCPI Party Y when it is able and willing to offer Party Issued Objects or answer remote procedure calls on behalf of X to Y. Note that the set of Party Issued Objects and Remote Procedure Calls that the Platform offers to Platform Y on behalf of Platform X does not have to be the same set that it offers to other OCPI Parties.
OCPI Base URL	A base URL from which all URLs to a Platform's OCPI endpoints can be derived by appending path segments and/or query parameters
OCPI Connection	The conceptual connection between two Platforms (in contrast with the technical HTTP connections used to actually exchange data). The Platforms have agreed that they will exchange information over OCPI. Platforms must keep some data about each OCPI Connection, for example for identification, authentication and to know what roles the other side has.
OCPI Party	An actor sharing data or requesting operations via OCPI. Parties are identified by a Party ID as defined in ISO 15118.
Platform Identity	A string identifying an OCPI Platform. When communicating using OCPI, Platforms use each other's Platform Identifier to identify the Platform that they are communicating with.

NOTE

The definition of Charge Token is deliberately abstract to include different kinds of Charge Tokens in practical use. Charge Tokens may have a physical form or they may purely be bits of information stored in computer systems. An RFID card issued by an MSP to operate Charging Stations is an example of a Charge Token with a physical form. Single-use authorization identifiers for Charging Sessions started by a Driver with a mobile phone application are examples of Charge Tokens without a physical form. Another example of what a Charge Token could be would be a bit of identifying information stored in the EV itself in order to enable a "Plug and Charge" experience for the Driver.

References

In the Functional Use Cases we will refer to these documents for normative descriptions of how OCPI platforms should behave:

Reference identifier	Title	URL
CSR	PKCS #10: Certification Request Syntax Specification, Version 1.7	https://datatracker.ietf.org/doc/html/rfc2986
HTTPMSG	HTTP/1.1: Message Syntax and Routing	https://datatracker.ietf.org/doc/html/rfc7230
JSON	ECMA-404 The JSON data interchange syntax	https://www.ecma-international.org/publications-and-standards/standards/ecma-404/
JSONSCHEMA	JSON Schema: A Media Type for Describing JSON Documents	https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-01
TLSGUID	IT Security Guidelines for Transport Layer Security (TLS), version 2.1	https://english.ncsc.nl/publications/publications/2021/january/19/it-security-guidelines-for-transport-layer-security-2.1
PEM	Textual Encodings of PKIX, PKCS, and CMS Structures	https://datatracker.ietf.org/doc/html/rfc7468
RFC2119	Key words for use in RFCs to Indicate Requirement Levels	https://datatracker.ietf.org/doc/html/rfc2119
RFC3339	Date and Time on the Internet: Timestamps	https://datatracker.ietf.org/doc/html/rfc3339/
RFC6848	Deprecating the "X-" Prefix and Similar Constructs in Application Protocols	https://datatracker.ietf.org/doc/html/rfc6648
RFC8174	Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words	https://datatracker.ietf.org/doc/html/rfc8174
TZVAL	Time Zone Database	http://www.iana.org/time-zones
URI	Uniform Resource Identifier (URI): Generic Syntax	https://datatracker.ietf.org/doc/html/rfc3986
X509	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	https://datatracker.ietf.org/doc/html/rfc5280

Use Case Template

This section contains the proposed Use Case template for OCPI 3.0. The number and name of the use case are not part of the table, but are in the header before the use case table.

UC: F01 - Example Use Case

Objective(s)	**1.** Objective 1 (optionally numbered) **2.** Objective 2 (but might also just contain a text.) **3.** Objective 3
Description	This contains a short description of the use case in words.
Actors	the actors involved in this use case: CPO, eMSP, Hub etc.
Flow	**1.** Step 1 **2.** Step 2 **3.** Step 3 **4.** Step 4 **5.** Step 5
Preconditions	Contains the pre-condition of the use case.
Postconditions	Contains the post-condition of the use case.
Error handling	Contains the error handling if applicable, error handling is not part of the requirements
Remark(s)	space for adding remarks to a use case.

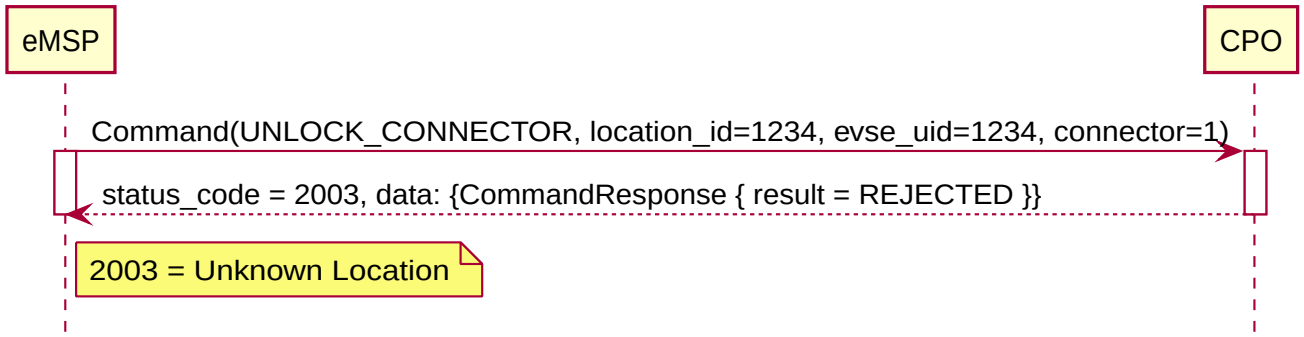


Figure 1. Sequence Diagram: Example Use Case

Table 1. UC: F01 Requirements

ID	Precondition	Requirement
R.F01.01	Pre condition of first requirement, if applicable	The first requirement itself.
R.F01.02	Pre condition of second requirement, if applicable	The second requirement itself.

Fundamental data type definitions

This chapter introduces fundamental data types that are used to define all data the types that are used in later chapters of this document.

class

When a data type is defined as a "class" in the OCPI specification, we mean a type whose possible values are sets of zero or more pairs of a string and another value. The string is known as a "key", "field name", or "property", and the value associated with the key is known as a field value. For each class type, the specification lists which strings are required and allowed to occur as field names in values of that type, and what the types of the field values of these fields should be.

In the serialized JSON form of OCPI messages, class values are serialized as JSON objects.

enum

When a data type is defined as an "enum" in the OCPI specification, we mean a type whose possible values are a finite number of strings.

This type is used for class fields where it is clear that there is only a finite set of possible values that is completely known at the time of writing of the specification. An example of a place where this is used is a class field whose possible values are the days of the week.

In the serialized JSON form of OCPI messages, enum values are serialized as JSON strings.

OpenEnum *type*

The OpenEnum type is meant for class fields for which the set of all possible values is not known at the time of writing of the specification, but where there are a finite number of known possible values. In this case we want to specify how OCPI implementers can use the known possible values, but also leave room for them to use other values.

This is used for example for connector types, where all implementers should use the same value to identify a widely used connector type like the Type 2 "Mennekes" plug, but where there should also be room for implementers to name new or custom plug types that were not taken into account by OCPI's authors.

In the serialized JSON form of OCPI messages, OpenEnum values are serialized as JSON strings.

When naming new OpenEnum values, OCPI implementers SHOULD follow the "Recommendations for Creators of New Parameters" found in IETF RFC 6848 ([RFC6848]).

UnicodeString *type*

A character string to be interpreted by human actors. OCPI poses no restrictions on the values of fields of this type beyond that they be valid JSON strings.

NOTE	This means that implementations must be prepared to receive arbitrary Unicode characters in fields with this type, including things that developers may not think of as text, like control characters,

> emoji and writing systems that are not their own.

When a size restriction is given with this type, it restricts the number of entries in the sequence of code points that makes up the string.

Size restrictions are given after the type name between square brackets. If there is one number between the square brackets, values have to have exactly this amount of entries in the code point sequence. If there are two numbers between the square brackets with ".." between them, the first number is the minimum code point sequence length, and the second number is the maximum code point sequence length.

So for example, for a type UnicodeString[3], "abc" and "💩🔥🚘" would be valid values, but "ab" and "abcdef" are not valid values. An emoji for a man with a certain skin tone with white hair is also not valid as a UnicodeString[3] because it consists of four code points, for "man", the skin tone, the zero-width joiner and the white hair respectively.

For a type UnicodeString[1..3], "a" and "💩🔥" and "abc" are valid values, but "abcdef" is not valid.

In the serialized JSON form of OCPI messages, UnicodeString values are serialized as JSON strings.

AsciiString *type*

A character string to be interpreted by information technology systems. Case sensitive. Only printable ASCII is allowed, that is all characters in these strings must have Unicode code points between U+0020 and U+007E inclusive.

Note that AsciiString allows spaces whereas CiAsciiString does not allow them.

When a size restriction is given with this type, it restricts the number of entries in the sequence of characters that makes up the string.

Size restrictions are given after the type name between square brackets. If there is one number between the square brackets, values have to have exactly this amount of entries in the character sequence. If there are two numbers between the square brackets with ".." between them, the first number is the minimum character sequence length, and the second number is the maximum character sequence length.

So for example, for a type AsciiString[3], "abc" would be a valid value, but "ab" and "abcdef" are not valid values. "éġç" and "αβγ" are also not valid values for AsciiString[3] because the characters in them are not printable ASCII.

For a type AsciiString[1..3], "a" and "abc" are valid values, but "abcdef" is not valid.

All strings in messages and enumerations are case sensitive, unless explicitly stated otherwise.

In the serialized JSON form of OCPI messages, AsciiString values are serialized as JSON strings.

CiAsciiString *type*

A character string to be interpreted by information technology systems. Case insensitive. Only printable ASCII excluding the space character is allowed, that is all characters in these strings must have Unicode code points between U+0021 and U+007E inclusive.

Note that CiAsciiString does not allow spaces whereas AsciiString allows them.

For size restrictions on CiAsciiString, the same rules as for AsciiString apply.

In the serialized JSON form of OCPI messages, CiAsciiString values are serialized as JSON strings.

int *type*

A non-negative integer number.

When a size restriction is given with this type, it restricts the number of decimal places.

Size restrictions are given after the type name between square brackets. If there is one number between the square brackets, values have to have exactly this amount of decimal places. If there are two numbers between the square brackets with ".." between them, the first number is the minimum size, and the second number is the maximum size.

So for example, for a type int[3], 123 would be a valid value, but 12 or 1234 are not valid values. For a type int[1..3], 12 and 123 are valid values, but 1234 is not valid.

In the serialized JSON form of OCPI messages, int values are serialized as JSON numbers.

decimal *type*

A number that is to be used with decimal arithmetic. These numbers may be positive or negative or zero. They may have as many decimals after the decimal point as is sufficient for correct price computation. Such numbers are used for calculating the cost of Charging Sessions.

When serializing and deserializing objects containing fields with a `decimal` type, and using the values contained in these objects, OCPI implementers have to take care to use arbitrary precision decimal arithmetic. This can be accomplished by using code libraries for this purpose. These may be built in to the language itself, like Java's `java.lang.BigDecimal` class, or third-party products like bignumber.js and dinero.js for Javascript and Typescript.

In the serialized JSON form of OCPI messages, decimal values are serialized as JSON numbers.

Using floating-point types to represent these values is *not* appropriate and will lead to incorrect price calculation. Unfortunately most JSON libraries will use floating point types to represent JSON numbers by default. Implementers have to configure them to use arbitrary-precision decimal types instead.

URL *type*

A string of at most 255 characters that follows the w3.org spec.

In the serialized JSON form of OCPI messages, URL values are serialized as JSON strings.

PartyID *type*

An ISO 15118 e-mobility Party ID. In OCPI payloads these must be encoded in their shortest form as 5-character strings, like "NLTNM".

Party IDs are case insensitive.

In the serialized JSON form of OCPI messages, PartyID values are serialized as JSON strings.

DateTime *type*

All timestamps SHALL be formatted as string following RFC 3339, with the following additional limitations:

- All timestamps SHALL be in UTC.
- Fractional seconds MAY be used.
- When fractional seconds are given, there SHALL be either one or three decimals after the point.

These are examples of valid values for *DateTime* fields:

```
2015-06-29T20:39:09Z
2016-12-29T17:45:09.2Z
2018-01-01T01:08:01.123Z
```

In the serialized JSON form of OCPI messages, DateTime values are serialized as JSON strings.

DisplayText *class*

Property	Type	Card.	Description
language	CiAsciiString[2]	1	Language Code ISO 639-1.
text	UnicodeString[0..512]	1	Text to be displayed to humans. Parties SHOULD NOT use HTML or other markup languages in this field.

Example:

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

1. Registration

This section contains the Functional Use Cases for Registration, that is the set-up of an OCPI connection between two OCPI Platforms.

1.1. Use Cases

1.1.1. UC: 01.01 - Initial credentials exchange (manual)

1	**Objective(s)**	**1.** Exchange credentials that allow either platform to initiate authenticated and confidential communication to the other platform
2	**Description**	Both platforms create a Certificate Signing Request (CSR), and send this to the other platform. They then obtain a client certificate from the partner platform that they can use to authenticate themselves to the other platform.
3	**Actors**	Any
4	**Flow**	**1.** Platform A creates a CSR **2.** Platform A sends that CSR to Platform B **3.** Platform B checks the CSR against OCPI's and Platform B's security requirements. OCPI's requirements are the requirements of this use case below. **4.** Platform B creates a client certificate based on the CSR that it received from Platform A **5.** Platform B sends this client certificate to Platform A
5	**Preconditions**	Both platforms have agreed to set up an OCPI connection between them
6	**Postconditions**	Platform A has a client certificate that allows it to make authenticated requests to Platform B
7	**Error handling**	When any step fails, retry the failing step or the whole procedure
8	**Remark(s)**	OCPI does not specify the technical means by which the CSRs and client certificates are to be exchanged. Typically one would use email or instant messaging applications. While this use case is described from the perspective of one platform, both platforms have to execute this use case in the initiating role in order to establish an OCPI connection.

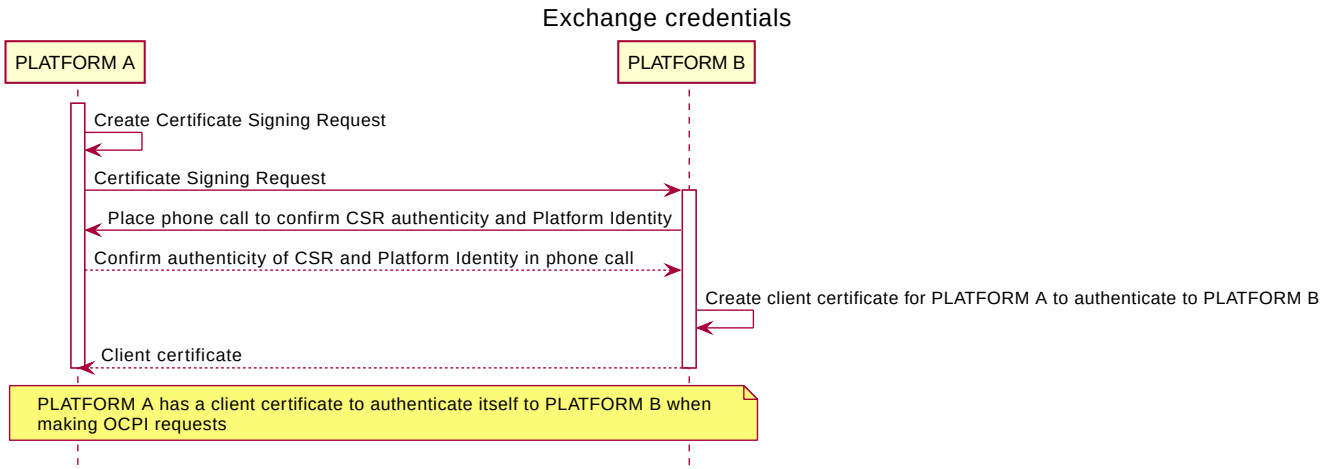


Figure 2. Sequence Diagram: Initial credentials exchange (manual)

Table 2. UC: 01.01 Requirements

ID	Precondition	Requirement
R.01.01.01		The commonName field of the CSR SHALL contain Platform A's Platform Identity
R.01.01.02		Platform A's Platform Identity SHALL be a domain name of which the platform creating the CSR controls the public DNS records
R.01.01.03		Platform A's Platform Identity SHOULD contain a domain name that is recognized by EV roaming professionals as identifying the platform creating the CSR
R.01.01.04		Platform B SHALL validate that the CSR indeed comes from Platform A by placing a phone call to Platform A and confirming the Platform Identity in the CSR with them
R.01.01.05		Cryptographic parameters at every step in the process SHALL be chosen following the guidelines in [TLSGUID]
R.01.01.06		Cryptographic parameters at every step in the process SHOULD be chosen in a way that is not "Phase out" according to [TLSGUID]
R.01.01.07		The CSR SHALL be restricted to TLS WWW Client Authentication and SHALL NOT be permitted for TLS WWW Server Authentication as described in [X509]

1.1.2. UC: 01.02 - Establish secure connection

1	**Objective(s)**	**1.** Set up a confidential and authenticated communication channel to another OCPI platform
2	**Description**	Platform A obtains a URL to an OCPI endpoint on Platform B. Platform A then sets up a TLS connection to the host from this URL.
3	**Actors**	Any
4	**Flow**	**1.** Platform A obtains a URL to one of its OCPI modules, typically by following a later use case from this document referencing this use case **2.** Platform A makes a request to the host from the URL to set up a TLS connection **3.** Platform A authenticates Platform B using a server certificate according to TLS **4.** Platform B authenticates Platform A using a client certificate according to TLS **5.** Both platforms agree on an encryption method according to TLS
5	**Preconditions**	Both platforms have exchanged credentials as per Initial credentials exchange
6	**Postconditions**	There is a confidential, authenticated communication channel through which Platform A can send its request to Platform B and Platform B can send its responses back
7	**Error handling**	If any step fails, platforms should check their credentials and their TLS parameters. If the credentials are found to be causing the problem, the platforms should perform Initial credentials exchange to obtain working credentials

8	**Remark(s)**	

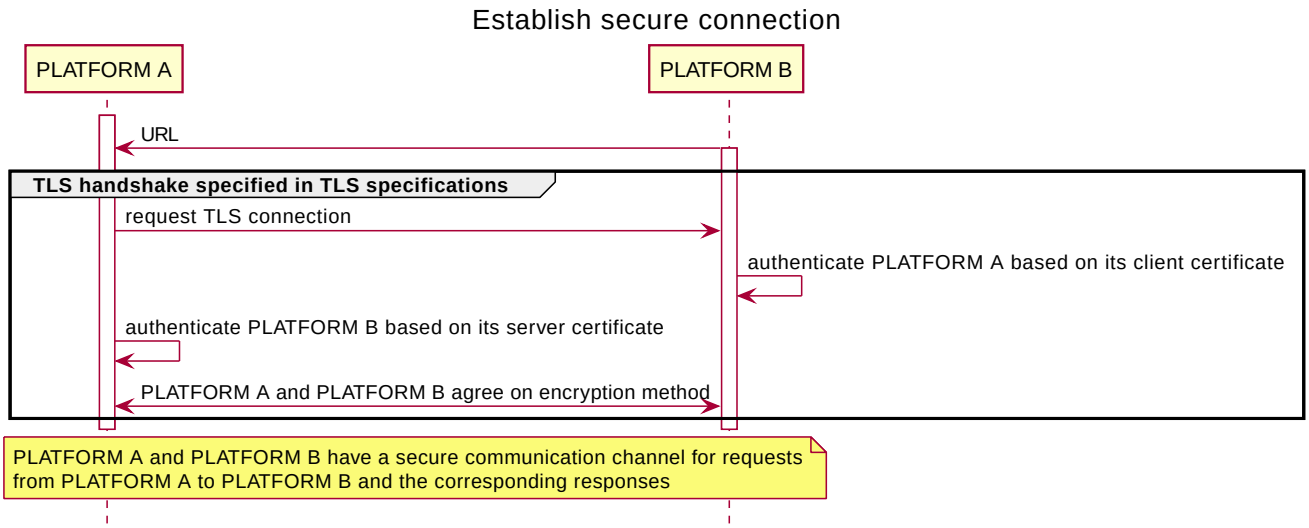


Figure 3. Sequence Diagram: Establish secure connection

Table 3. UC: 01.02 Requirements

ID	Precondition	Requirement
R.01.02.01		The URL used by Platform A SHALL be an "https" scheme URL according to [HTTPMSG]
R.01.02.02		PLATFORM A SHALL validate that PLATFORM B presents a valid server certificate following the guidelines in [TLSGUID]
R.01.02.03		PLATFORM B SHALL validate that PLATFORM A presents a valid client certificate as produced by PLATFORM B according to Initial credentials exchange
R.01.02.04		The platforms SHALL enforce the guidelines in [TLSGUID] during the TLS handshake
R.01.02.05		The platforms SHOULD NOT allow a TLS connection to be set up with parameter choices that are "Phase out" according to [TLSGUID]

1.1.3. UC: 01.03 - Handshake OCPI connection parameters

1	**Objective(s)**	**1.** Set the OCPI version to use and exchange OCPI base URLs between the partner platforms
2	**Description**	Platform B shares its OCPI base URL with Platform A. Platform A then sends a POST request with its desired OCPI version, OCPI base URL and request timeout to Platform B. Platform B responds to indicate its acceptance of the connection, also giving its own request timeout in the response body.
3	**Actors**	Any

4	**Flow**	**1.** Platform B shares its OCPI base URL with Platform A **2.** Platform A sets up a secure communication channel to Platform B according to Establish secure connection **3.** Platform A makes an HTTP POST request to Platform B's base URL with "/connection" appended over the secure channel. The request body contains Platform A's desired OCPI version **4.** Platform B responds to this request with its own parameters
5	**Preconditions**	Both platforms have exchanged credentials as per Initial credentials exchange
6	**Postconditions**	The two platforms have an OCPI connection between them.
7	**Error handling**	If Platform B already has a connection to Platform A, it SHOULD respond to the incoming connection POST request with HTTP status 409 Conflict If Platform B does not wish to set up an OCPI connection at the OCPI version that Platform A requests, it should respond to the incoming connection POST request with an HTTP status 409 Conflict
8	**Remark(s)**	OCPI does not specify the technical means by which Platform B's base URL is shared with Platform A. Typically one would use email or instant messaging applications.

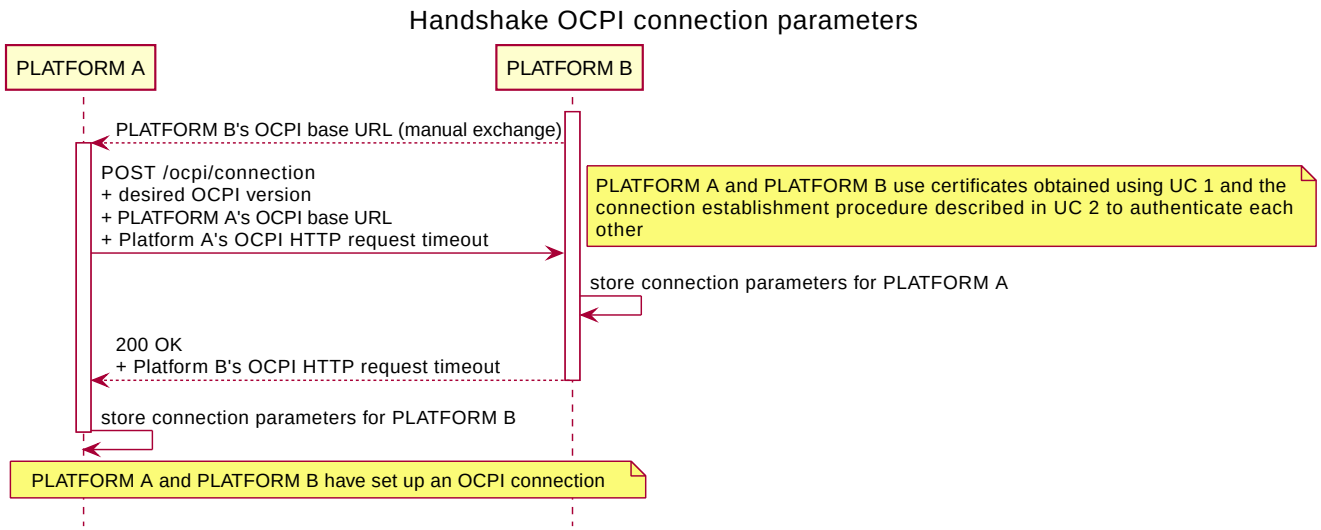


Figure 4. Sequence Diagram: Handshake OCPI connection parameters

Table 4. UC: 01.03 Requirements

ID	Precondition	Requirement
R.01.03.01		The OCPI base URL for Platform B SHALL be an "https" scheme URL according to [HTTPMSG]
R.01.03.02		Platform A SHALL set up the connection to Platform B according to Establish secure connection
R.01.03.03		Platform A SHALL obtain the URL to make the POST to by appending "/connection" to the path of the OCPI base URL of Platform B as obtained in Handshake OCPI connection parameters
R.01.03.04		Platform A SHOULD put a ConnectionParametersRequest object in the request body

ID	Precondition	Requirement
R.01.03.05		Platform A SHOULD set its requested version in the ConnectionParametersRequest object to "3.0"
R.01.03.06	Platform B intends to accept the connection handshake offer from Platform A	Platform B SHOULD respond with a success status code
R.01.03.07	Platform B intends to accept the connection handshake offer from Platform A	Platform B SHOULD put a ConnectionParametersResponse object in the response body
R.01.03.08	Platform B already has an OCPI connection to Platform A	Platform B SHOULD respond to the incoming POST request with HTTP status 409 Conflict
R.01.03.09	Platform B does not wish to set up an OCPI connection with Platform A at the requested version	Platform B SHOULD respond to the incoming POST request with HTTP status 409 Conflict

1.1.4. UC: 01.04 - Renew certificate

1	**Objective(s)**	**1.** Platform A obtains a new client certificate to make authenticated requests to Platform B
2	**Description**	Certificates for use with TLS have a limited validity period. By renewing their client certificate platforms can make sure that their requests to a partner platform can be authenticated after the validity of their current client certificate expires.
3	**Actors**	Any
4	**Flow**	**1.** Platform A makes an HTTP POST request to Platform B over a secure connection set up according to Establish secure connection, giving Platform B a CSR and a callback ID with which Platform B can post the client certificate **2.** Platform B responds with HTTP status code 200 OK **3.** Platform B checks if the CSR meets OCPI's and Platform B's requirements **3.** Platform B creates a new client certificate for Platform A **4.** Platform B makes an HTTP POST request to a URL containing the callback ID given by Platform A at step 1 with the client certificate in the request body **5.** Platform A starts using the new client certificate for its requests to Platform B
5	**Preconditions**	Both platforms have exchanged client certificates already according to Initial credentials exchange. Both these client certificates are valid at the time this use case is attempted.
6	**Postconditions**	Platform A is using a new client certificate to make authenticated requests to Platform B
7	**Error handling**	If Platform B is unable or unwilling to create a new client certificate for Platform A, it makes an HTTP POST request to the same URL that it would post the new certificate to, indicating the reason for not providing a client certificate in the request body. If one of the platforms does not have a valid client certificate to authenticate to the other one, it should obtain a valid client certificate with Initial credentials exchange before attempting this use case.
8	**Remark(s)**	This use case is specified so that Platform B is free to choose a manual or a fully automated process for issuing the new client certificate

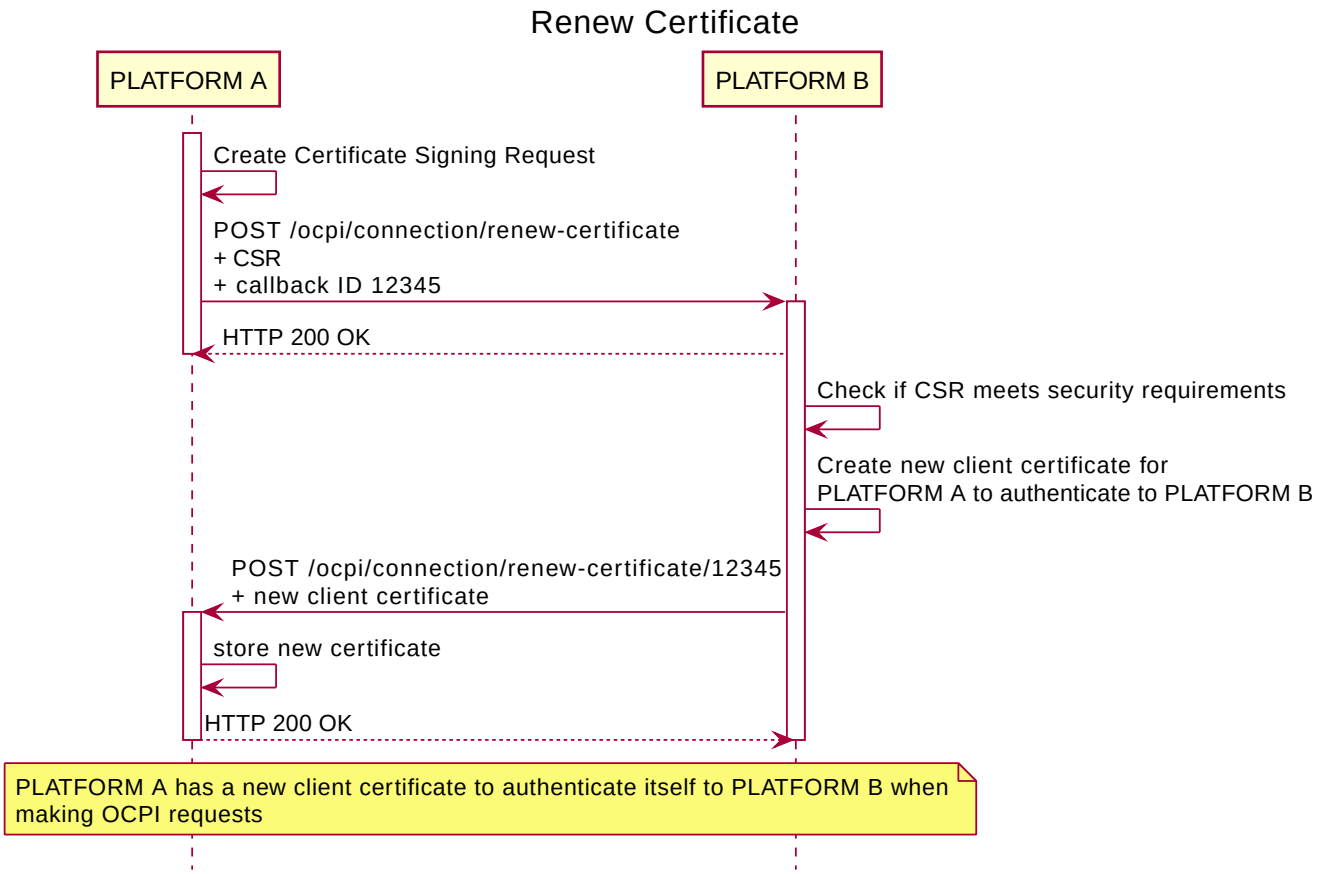


Figure 5. Sequence Diagram: Renew certificate

Table 5. UC: 01.04 Requirements

ID	Precondition	Requirement
R.01.04.01		Platform A SHALL obtain the URL to make the POST request with the CSR to by appending "/connection/renew-credentials" to the path of the OCPI base URL of Platform B as obtained in Handshake OCPI connection parameters
R.01.04.02		Platform A SHALL set up the connection to Platform B according to Establish secure connection
R.01.04.03		Platform A SHALL send a CertificateRenewalRequest object in the request body of the POST request to Platform B
R.01.04.04	Platform B received a CertificateRenewalRequest object from Platform A	Platform B SHOULD make its POST request with either a certificate or a reason for rejection within 168 hours
R.01.04.05		Platform B SHALL check if the value of the commonName field in the CSR submitted by Platform A is the same as the one in the commonName field of the certificate that Platform A is using to authenticate to Platform B

ID	Precondition	Requirement
R.01.04.06	Platform B finds that the value of the commonName field in the CSR is different from the value of the commonName field of the certificate that Platform A is using to authenticate to Platform B	Platform B SHALL NOT issue a new certificate to Platform A
R.01.04.07	Platform B issued a new certificate to Platform A	Platform B SHOULD make the POST request to Platform A to a URL obtained by appending `/connection/renew-credentials/` and then appending the callback ID given by Platform A to Platform A's OCPI base URL
R.01.04.08	Platform B issued a new certificate to Platform A	Platform B SHALL send a RenewedCertificate object in the request body of the POST request to Platform A
R.01.04.09	Platform B decided to not issue a new certificate to Platform A	Platform B SHOULD make a POST request to Platform A to a URL obtained by appending `/connection/renew-credentials/` and then appending the callback ID given by Platform A to Platform A's OCPI base URL
R.01.04.10	Platform B decided to not issue a new certificate to Platform A	Platform B SHOULD give the reason for not issuing a new certificate to Platform A in the request body of its POST request to Platform A

1.1.5. UC: 01.05 - Terminate OCPI connection

1	**Objective(s)**	**1.** Stop exchanging data and requests between platforms in a graceful way so that both platforms are aware that the connection is terminated
2	**Description**	Platform A sends a request to terminate the OCPI connection to Platform B. Platform B acknowledges this and both platforms stop accepting each other's client certificates.
3	**Actors**	Any
4	**Flow**	**1.** Platform A sends a "bye" request to Platform B **2.** Platform B stops accepting the client certificate of Platform A for new requests **3.** Platform B responds to the "bye" request **4.** Platform A stops accepting the client certificate of Platform B for new requests
5	**Preconditions**	Platform A has a client certificate to authenticate to Platform B, obtained by Initial credentials exchange or Renew certificate.
6	**Postconditions**	There is no OCPI connection between the platforms
7	**Error handling**	n/a
8	**Remark(s)**	n/a

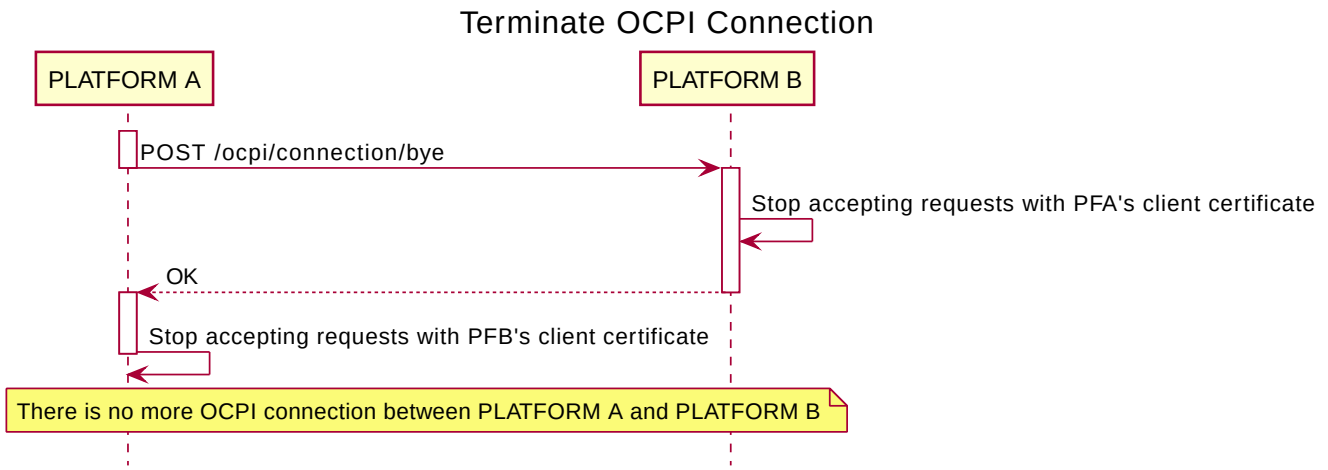


Figure 6. Sequence Diagram: Terminate OCPI connection

Table 6. UC: 01.05 Requirements

ID	Precondition	Requirement
R.01.05.01		Platform A SHALL obtain the URL to make the POST request for termination to by appending "/connection/bye" to the path of the OCPI base URL of Platform B as obtained in Handshake OCPI connection parameters
R.01.05.02		Platform A SHALL set up the connection to Platform B according to Establish secure connection
R.01.05.03		After Platform B has responded to Platform A, Platform B SHOULD NOT accept incoming requests with any client certificate with the commonName field set to the Platform Identity of Platform A
R.01.05.04		After Platform A has processed the response from Platform B, Platform A SHOULD NOT accept incoming requests with any client certificate with the commonName field set to the Platform Identity of Platform B

1.1.6. UC: 01.06 - Request supported OCPI versions

1	**Objective(s)**	**1.** Learn which versions of OCPI are supported by a partner platform
2	**Description**	Platform A makes a request to Platform B for Platform B's supported OCPI versions. Platform B responds with a list of supported versions and the base URLs to use for them.
3	**Actors**	Any
4	**Flow**	**1.** Platform A requests supported OCPI versions from Platform B **2.** Platform B responds to Platform A with a list of OCPI versions it supports
5	**Preconditions**	Platform A has a client certificate to authenticate to Platform B, obtained by Initial credentials exchange or Renew certificate.

6	**Postconditions**	Platform A has information on which OCPI versions Platform B supports
7	**Error handling**	n/a
8	**Remark(s)**	Unlike in previous OCPI versions, requesting the list of supported OCPI versions is not a necessary step in setting up an OCPI connection between two platforms

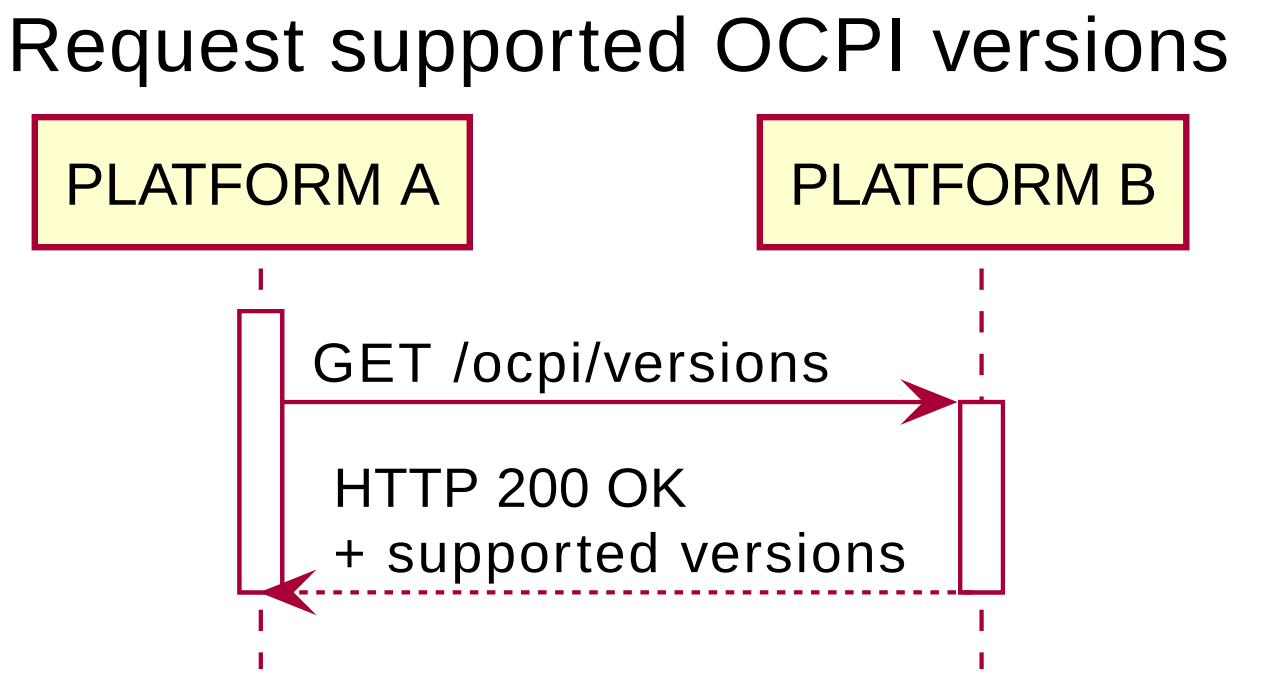


Figure 7. Sequence Diagram: Request supported OCPI versions

Table 7. UC: 01.06 Requirements

ID	Precondition	Requirement
R.01.06.01		Platform A SHALL obtain the URL to make the GET request to by appending "/versions" to the path of the OCPI base URL of Platform B as obtained in Handshake OCPI connection parameters
R.01.06.02		Platform A SHALL set up the connection to Platform B according to Establish secure connection
R.01.06.03		The response body of Platform B's response SHOULD contain an OcpiVersions object

1.2. Object types for Registration use cases

1.2.1. ConnectionParametersRequest *class*

Field Name	Type	Cardinality	Description

version	AsciiString[1..12]	1	The OCPI version that Platform B requests to use with Platform A
request_timeout	integer	1	A timeout, in milliseconds, for Platform A to use when making OCPI HTTP requests to platform B. That is, after this many milliseconds have elapsed after Platform A made a request without Platform A receiving a response to it, Platform A SHOULD not expect to receive a response from Platform B to that request anymore.
base_url	URL	1	Platform B's base URL

1.2.2. ConnectionParametersResponse *class*

Field Name	Type	Cardinality	Description
request_timeout	integer	1	A timeout, in milliseconds, for Platform B to use when making OCPI HTTP requests to platform A. That is, after this many milliseconds have elapsed after Platform B made a request without Platform B receiving a response to it, Platform B SHOULD not expect to receive a response from Platform A anymore.

1.2.3. CertificateRenewalRequest *class*

Field Name	Type	Cardinality	Description
csr	AsciiString[0..5500]	1	The certificate signing request according to RFC 2986 ([CSR]), encoded according to PEM ([PEM])
callbackId	AsciiString[0..36]	1	The callback ID for the other party to create a URL to post the signed certificate.

1.2.4. OcpiVersions *class*

Field Name	Type	Cardinality	Description
versions	OcpiVersion	+	The versions of OCPI that the platform supports

1.2.5. OcpiVersion *class*

Field Name	Type	Cardinality	Description
version	AsciiString[3..10]	1	The version number of the OCPI version being described in this record
url	URL	1	The OCPI base URL for this platform for version 3.0. For earlier versions of OCPI, the URL to the endpoint containing version specific information.

1.2.6. RenewedCertificate *class*

Field Name	Type	Cardinality	Description
certificateChain	AsciiString[0..10000]	1	The certificate chain for the receiving platform to identify itself. The certificate SHOULD be a X.509 certificate ([X509]), encoded according to PEM ([PEM]). The PEM bundle MAY also contain sub CA certificates. In case it does, the certificates SHOULD be ordered in the bundle from the leaf certificate to the root certificate.

2. Request Addressing

This section contains the Functional Use Cases for the Addressing of OCPI requests.

2.1. Note from the editor on changes from OCPI 2.2

This section is now addressed to reviewers of OCPI 3.0 drafts. It may be removed or edited into a non-normative section for OCPI implementers after reviews

This "request addressing" chapter is the outcome of looking at the Roaming Hub related Business Use Cases gathered for OCPI 3.0.

Compared to OCPI 2.2, OCPI 3.0 takes a simpler approach to addressing requests to Parties on Platforms operated by Roaming Hubs or other kinds of platforms that serve multiple Parties.

OCPI 3.0 does not have any data types or request-response flows that are specific to Roaming Hubs or other complex Platform topologies. Instead, OCPI 3.0 aims to provide basic data types and use cases about making requests between Parties hosted on Platforms. These message flows and data types are intended to be flexible enough to be used by simple and complex Platforms alike.

This makes OCPI simpler, because the things that make different kinds of Platforms different from each other are no longer OCPI's concern. And the simpler OCPI is, the quicker it can be implemented and the more interoperability it offers between implementing Platforms. It does mean that the OCPI use cases are more abstract and may take some more effort to be understood in the context of a specific type of Platform.

I have tried to offer specific examples for certain types of Platform topologies where I thought this was in order. Please provide feedback in your review on how clear the consequences of this rework of Hub support are for you, and what we could do to make it easier to understand.

If you wonder where the "GET ALL" and "Broadcast Push" from OCPI 2.2 and 2.2.1 have gone: those will be taken care of in a subsequent chapter about a Pub/sub mechanism aka subscription model aka Party Issued Object replication. The basic idea is that now that we have Hub Party IDs and a way to address requests to Parties, you can subscribe to a Hub Party to "GET ALL", or a Hub Party can subscribe to a Party's data to do a reverse "Broadcast Push". The Broadcast Push has to be reverse because in OCPI 3.0, all data replication will be initiated by the data consumer, never the data producer.

While working on these Request Addressing use cases, I have also made a bunch of other changes that I stumbled upon:

- The polarities of Platform A and Platform B were reserved in the Establish Secure Connection and Handshake OCPI Connection Parameters use cases.
- I replaced the String and CiString datatypes by AsciiString, CiAsciiString and UnicodeString to allow non-ASCII text where this is in order.
- Instead of separate country codes and party ID fields, all messages referring to parties now use a single 5-character Party ID field, as suggested by Rudolph Froger.
- Contact information for technical matters was added to the metadata exchanged about Parties, as suggested in OCPI Development meetings.

2.2. Introduction

OCPI is a network protocol that is used by software systems called *Platforms* to transfer data and procedure calls between *Parties*. Therefore, when making an OCPI request, a Platform has to specify in the request which Party is the sender and which Party is the intended recipient. The use cases in this chapter describe how Platforms should do this.

Where the Use Cases in this chapter refer to "Hub" or "Roaming Hub", the Roaming Hub role is meant as defined in the Business Use Cases document, section 1.2.2, EV Charging Market Roles.

2.3. Use Cases

2.3.1. UC: 02.01 - Request Parties served by Platform

1	**Objective(s)**	**1.** Platform A learns which parties are available for OCPI connection through Platform B
2	**Description**	Platform A obtains a list of parties served from Platform B
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B. 2. Platform B responds with the list of parties that it serves to Platform A.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters.
6	**Postconditions**	Platform A knows which Parties Platform B serves to it.
7	**Error handling**	n/a
8	**Remark(s)**	OCPI 2.2.1 and 2.2 distinguished "roles" exchanged using the credentials module and "client info" exchanged using the hub client info module. OCPI 3.0 does not distinguish between the use cases behind those two mechanisms. Platform A reports all of the Parties it serves to Platform B in this use case, regardless of which relation these Parties have to Platform A. Typically, Platform A will want to periodically execute this use case flow to check for changes in the set of Parties that Platform B is serving to Platform A.

Figure 8. Sequence Diagram: Request Parties served by Platform

Table 8. UC: 02.01 Requirements

ID	Precondition	Requirement
R.02.01.01		Platform A SHALL obtain the URL to make the GET request to by appending "/parties" to the path of the OCPI base URL of Platform B as obtained in Handshake OCPI connection parameters
R.02.01.03		Platform A SHALL set up the connection to Platform B according to Establish secure connection
R.02.01.02		The response body of Platform B's response SHOULD contain a PlatformParties object
R.02.01.03	Platform B offers Hub functionalities to Platform A	Platform B SHALL fill in the party ID of the Hub party on its platform in the `hub_party_id` field of its response
R.02.01.04	Platform B offers no Hub functionalities to Platform A	Platform B SHALL leave the `hub_party_id` field unset in its response

2.3.2. UC: 02.02 - Make a request on behalf of a Party to a Party on another Platform

1	**Objective(s)**	**1.** A request from a Party to a Party is relayed over a connection from a Platform to a Platform
2	**Description**	Request and response messages are enveloped with addressing fields in order to allow party-to-party requests to be exchanged over a platform-to-platform connection
3	**Actors**	Any

4	**Flow**	1. Platform A adds addressing information to a request from Party X to Party Y by putting the sender and receiver Party IDs in the URL 2. Platform A sends the request to Platform B 3. Platform B handles the request on behalf of Party Y 4. Platform B envelopes the response data with addressing fields indicating the answering party and the intended receiver party 5. Platform B sends the response to Platform A 6. Platform A handles the enclosed response on behalf of Party X
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform A serves Party X to Platform B. Platform B serves Party Y to Platform A. Platform A has a URL and an HTTP request verb for an operation that it wants to request on behalf of Party X from Party Y NOTE: URLs and HTTP request verbs for operations are given in this document in later use cases that reference this use case.
6	**Postconditions**	Party X's request has been answered by Platform B on behalf of Party Y
7	**Error handling**	If Platform B cannot answer the request on behalf of Party Y, it can send a response to inform Platform A and Party X of this
8	**Remark(s)**	OCPI does not impose requirements on how Platform B communicates with Party Y to come up with a response on behalf of Party Y. Two different approaches are possible: local lookup and request forwarding. Local lookup will typically be used by Platforms commissioned to host a single Party, like where a CPO is running a Platform of their own to integrate with other Parties. Roaming Hubs may also use local lookup, exchanging the information needed to answer requests on behalf of Party Y with Party Y before a request from Party X has come in yet. The alternative approach of request forwarding is typically used by Roaming Hubs. Here, upon receiving the request on behalf of Party X from Platform A, Platform B forwards this request in some way to another system that can answer it on behalf of Party Y, and then relays the response to Platform A again using OCPI. In this scenario, the communication channel between Platform B and the other system may or not be an OCPI connection; whatever it is is not of concern to the OCPI Connection between Platform A and Platform B. Extra sequence diagrams are given below to clarify the local lookup and request forwarding variants. OCPI 2.2 and 2.2.1 allowed the use of HTTP headers for addressing on the response messages. OCPI 3.0 has no equivalent functionality. This is because the request and response messages are already correlated by the HTTP protocol. When a Platform receives a response, it knows that the sender of this response should be the receiver of the corresponding request and the receiver should be the sender of the response. Allowing Platforms to also provide their own addressing headers presents an unnecessary security risk as a malicious Platform could spoof OCPI's addressing headers while they cannot spoof the HTTP-level request-response correlation.

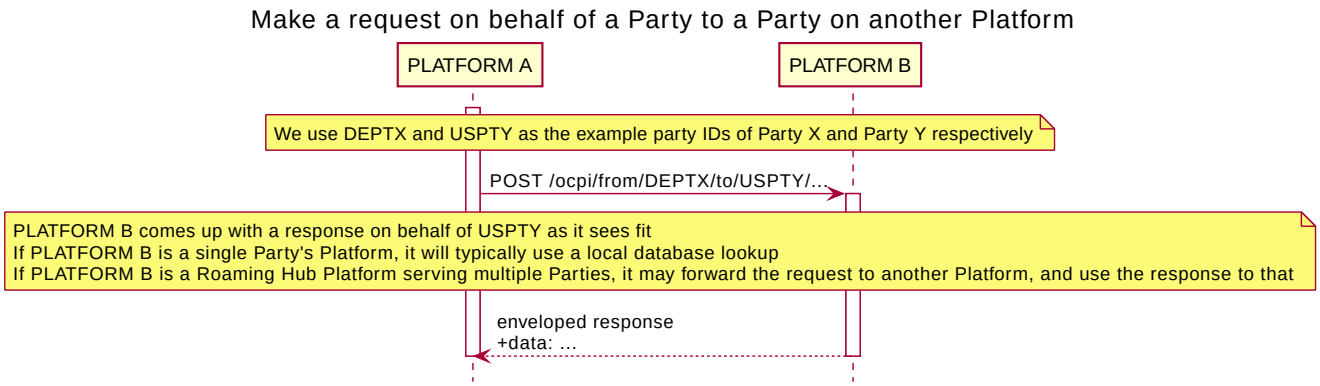


Figure 9. Sequence Diagram: Make a request on behalf of a Party to a Party on another Platform

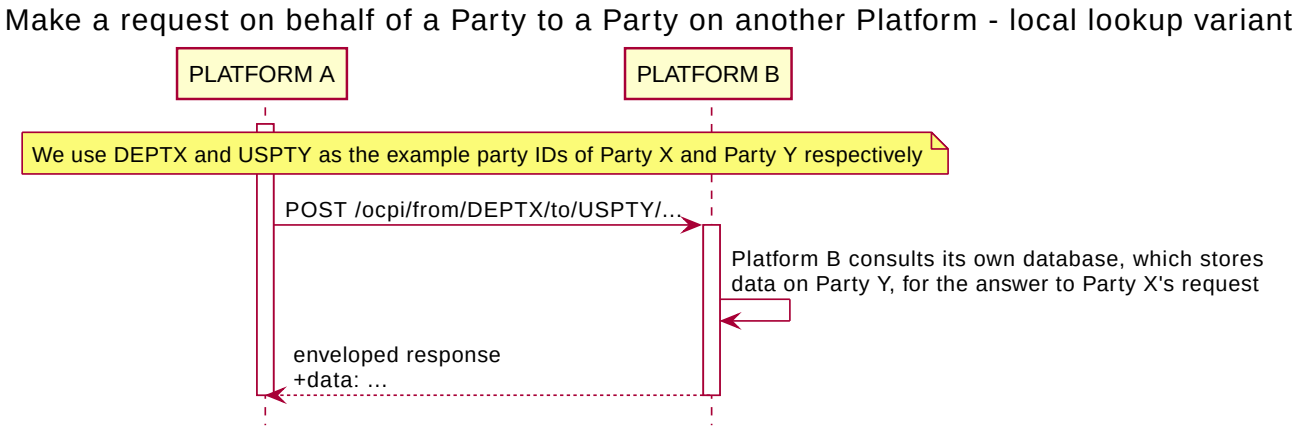


Figure 10. Sequence Diagram: Make a request on behalf of a Party to a Party on another Platform - local lookup variant

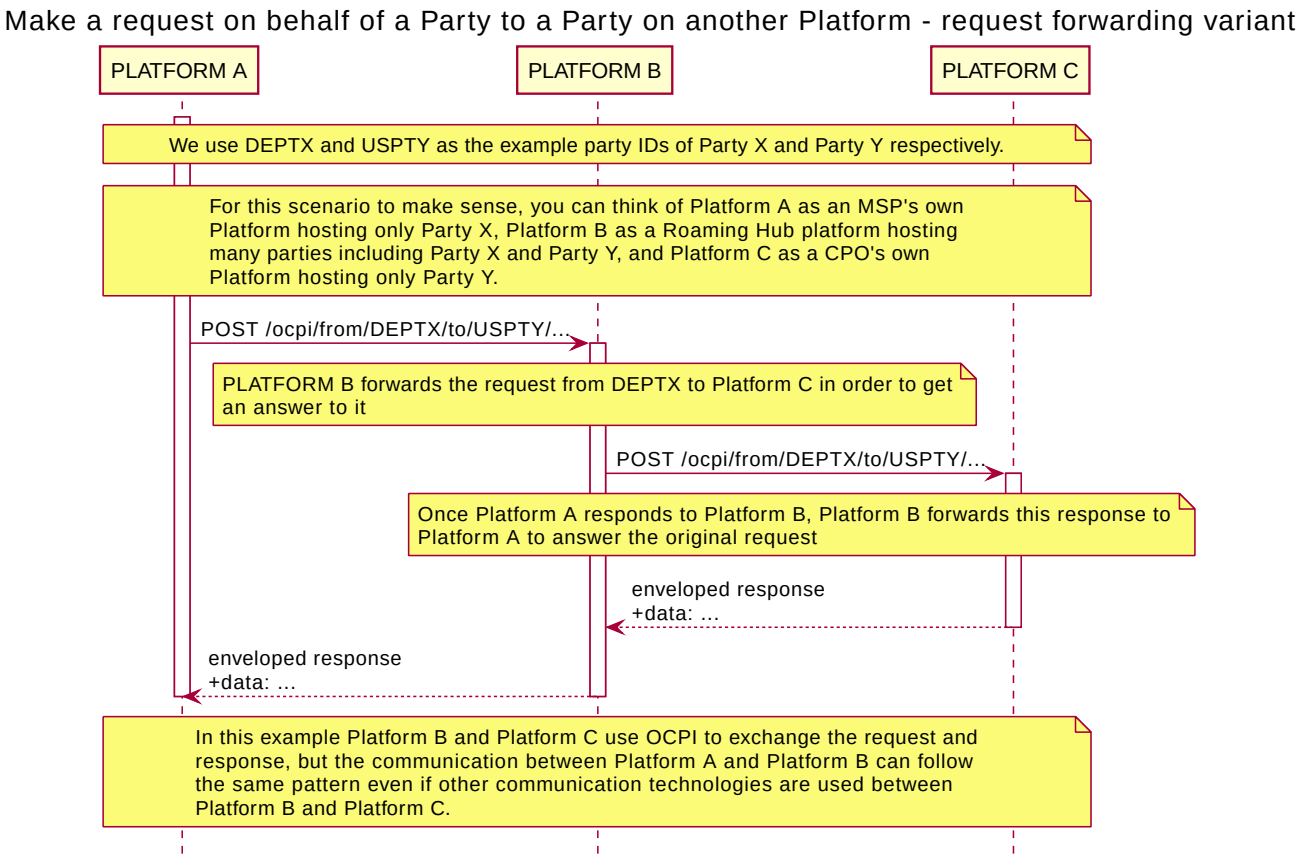


Figure 11. Sequence Diagram: Make a request on behalf of a Party to a Party on another Platform - request forwarding variant

Table 9. UC: 02.02 Requirements

ID	Precondition	Requirement
R.02.02.01		The URL that Platform A makes a request to SHALL NOT differ from the OCPI base URL of Platform B in other components than path, query and fragment as defined in [URI]
R.02.02.02		The URL that Platform A makes a request to SHALL have a path component that is obtained by adding the following things to the base URL in the order they are presented here: a segment "from", a segment that is the five-character Party ID of Party X that is the sender, a segment "to", a segment that is the five-character Party ID of Party Y that is the receiver, and finally the relative path specified in a later use case referencing this use case
R.02.02.03		Platform A SHALL set up the connection to Platform B according to Establish secure connection
R.02.02.04		The response body of Platform B's response to Platform A SHALL contain an OcpiResponse object

ID	Precondition	Requirement
R.02.02.05	When Platform B receives a request with a body that is not a valid JSON object according to [JSON]	Platform B MUST respond with an HTTP response with status code 400 (Bad Request)
R.02.02.06	When Platform B receives a request with a body that is syntactically valid JSON and addresses an existing resource	Platform B MUST NOT respond with an HTTP response with an HTTP status code indicating a client error
R.02.02.07	When Platform B receives a GET request for a resource that does not exist	Platform B SHOULD respond with an HTTP response with status code 404 (Not Found)
R.02.02.08	When Platform B receives a request with a body containing a valid OCPI object, and the object already existed, and the object has been successfully updated by Platform B	Platform B SHOULD respond with an HTTP response with status code 200 (OK)
R.02.02.09	When Platform B receives a request with a body containing a valid OCPI object and the object has been newly created in Platform B's ssystem as a result of the request	Platform B SHOULD respond with an HTTP response with status code 201 (Created)
R.02.02.10	When Platform B receives a request with a body containing a valid OCPI object but is not able to update or create a resource	Platform B SHOULD respond with an HTTP response with status code 200 (OK) and set the `status` field of the response body to one of the error statuses documented under Status Codes
R.02.02.11	When Platform B sends a response to a request that is made by Platform A according to a use case described in this document	Platform B SHALL NOT use custom status code range values in the `status` field of the response body
R.02.02.12		Platform A SHOULD set an HTTP header `Ocpi-Request-Id` on its request to Platform B with a value that is a newly generated UUID
R.02.02.13	Platform A is making its request to Platform B in order to answer another OCPI request that Platform A received, as in the Request Forwarding flow	Platform A SHOULD set an HTTP header `Ocpi-Correlation-Id` on its request to Platform B with a value that is the same as the value of the `Ocpi-Correlation-Id` header in the request that Platform A received
R.02.02.14	Platform A is not making its request to Platform B in order to answer another request that Platform A received	Platform A SHOULD set an HTTP header `Ocpi-Correlation-Id` on its request to Platform B with a value that is a newly generated UUID
R.02.02.15		Platform B SHOULD set an HTTP header `Ocpi-Request-Id` on its response to Platform A with a value that is the same as the value of the `Ocpi-Request-Id` header in the request that Platform B received
R.02.02.16		Platform B SHOULD set an HTTP header `Ocpi-Correlation-Id` on its response to Platform A with a value that is the same as the value of the `Ocpi-Correlation-Id` header in the request that Platform B received

NOTE While R.02.02.11 prohibits the use of custom status codes in responses to requests following use cases in this document, custom status codes can be used when Platforms are using custom extensions to request operations or exchange data that are not in scope of this document.

NOTE When custom status codes are used, keep in mind that different custom modules could use the same values with a different meaning, as they are not standardized.

NOTE the `Oci-Request-Id` and `Ocpi-Correlation-Id` serve to facilitate debugging and to allow detection of routing loops in complex topologies. When requests are proxied, the `Ocpi-Request-Id` changes on each hop, whereas the `Ocpi-Correlation-Id` stays the same, as in the diagram below.

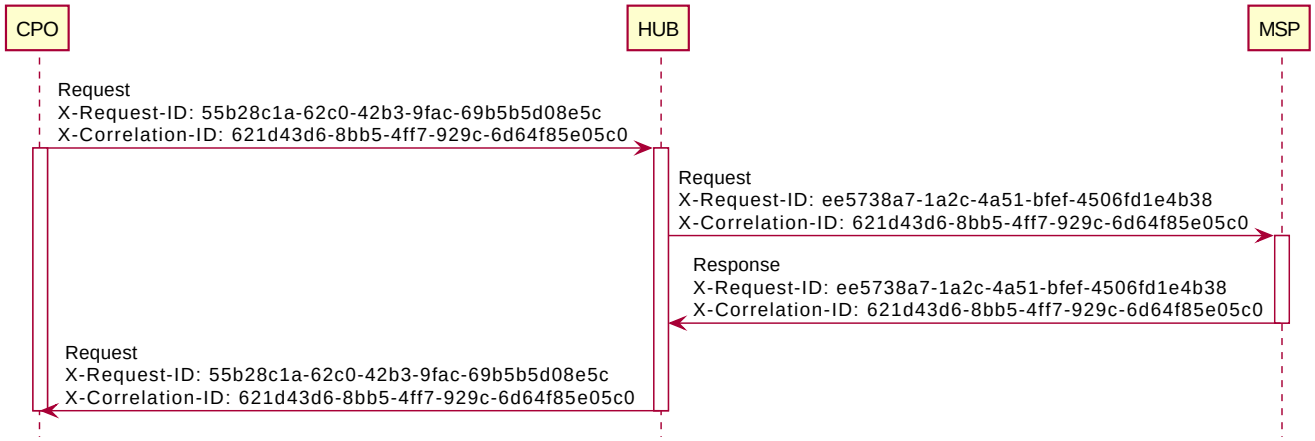


Figure 12. Sequence Diagram: Make a request on behalf of a Party to a Party on another Platform - Ocpi-Request-Id and Ocpi-Correlation-Id

2.4. Object types for Request Addressing use cases

2.4.1. BusinessDetails *class*

Gives more information about an organization that is a participant in OCPI communication as a Location operator, a Party or a Platform.

Property	Type	Card.	Description
name	UnicodeString[1..100]	1	Name of the company.
website	URL	?	URL of the company's website.
logo	Image	?	The company's logo.
technical_contact	PointOfContact	?	Contact point of the company for technical matters.

2.4.2. Image *class*

This class references an image related to an EVSE in terms of a file name or url. According to the roaming connection between one EVSE Operator and one or more Navigation Service Providers, the hosting or file exchange of image payload data has to be defined. The exchange of this content data is out of scope of OCPI. However, the

recommended setup is a public available web server hosted and updated by the EVSE Operator. Per Charging Station an unlimited number of images of each type is allowed. Recommended are at least two images where one is a network or provider logo and the second is a station photo. If two images of the same type are defined, not only one should be selected but both should be displayed together.

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

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

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

2.4.3. 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 views.
OPERATOR	Logo of the Charge Point Operator, for example a municipality, to be displayed in the EVSEs detailed information view or in lists and maps, if no network logo is present.
OTHER	Other
OWNER	Logo of the Charging Station owner, for example a local store, to be displayed in the EVSEs detailed information view.

2.4.4. InterfaceRole *enum*

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

2.4.5. ModuleID *OpenEnum*

Identifiers for OCPI modules.

An OCPI module is a type of data that can be exchanged between EV charging market parties via OCPI with its associated specifications.

Each modules' specific datatypes and specifications are described in its own chapter in this document.

The following table contains the list of modules in this version of OCPI. Modules are optional in the sense that an OCPI implementation does not have to offer each module. There may be dependencies between modules however, that is, it may be the case that some modules cannot be offered without also offering one or more other modules. If there are such dependencies between modules, it will be mentioned in the affected modules' descriptions.

Module	ModuleID
CDRs	`cdrs`
ChargingProfiles	`chargingprofiles`
Commands	`commands`
EVSE Status	`evsestatuses`
Invoice Reconciliation	`irrs`
Locations	`locations`
Sessions	`sessions`
Power Regulation	`meterreadings`
Tariffs	`tariffs`
Tariff Assocations	`tariffassociations`
Tokens	`tokens`

2.4.6. OcpiResponse *class*

Field Name	Type	Cardinality	Description
routed_receiver	CiAsciiString[5]	?	The party ID of the party that a Roaming Hub is answering the request on behalf of, if any

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

2.4.7. OCPI response status codes

OCPI defines its own set of status codes, apart from the ones defined by HTTP. The use of these status codes is described in the Make a request to a party on behalf of a party use case.

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
4xxx	Hub errors - The roaming hub ran into a problem routing the request to the receiver party
5xxx	Subscription errors - A Party Issued Object subscription cannot be created or torn down
6xxx	Party Issued Object errors - The update cannot be applied to the replicated Party Issued Object
7xxx	Remote Procedure Call errors - The requested operation resulted in an error

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

2.4.7.1. 1xxx: Success

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

2.4.7.2. 2xxx: Client errors

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

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

2.4.7.3. 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
39xx	Reserved range for custom server error status codes (3900-3999).

2.4.7.4. 4xxx: Hub errors

For errors that a Platform encounters when sourcing information from a request receiver Party, the following OCPI status codes SHALL be used.

Code	Description
4000	Generic error
4001	Unknown receiver (The receiver Party ID in the URL is not recognized or not available for making requests to)
4002	Timeout on forwarded request (message is forwarded, but request times out)
4003	Connection problem (Communication to an external system was needed to answer the request, but this communication failed)
49xx	Reserved range for custom hub error status codes (4900-4999).

2.4.7.5. 5xxx: Subscription errors

For errors that can occur when setting up or tearing down subscriptions, the following OCPI status codes SHALL be used.

Code	Description
5000	Generic error
5001	Subscriber party not recognized
5002	Subscriber party not authorized
5003	The party being subscribed to is not served by the Platform being subscribed to
5004	The module being subscribed to is not served for the party being subscribed to by the Platform being subscribed to
5005	No such subscription
5006	No such object

2.4.7.6. 6xxx: Platform Issued Object update errors

For errors that can occur when applying updates to a Party Issued Object, the following OCPI status codes SHALL be used.

Code	Description
6000	Generic error
6001	Schema violation (the PIO after update does not conform to the data schema for the module)
6002	No such subscription
6003	Immutable object (attempt to update an object to a new version while the module declares it as immutable)

2.4.7.7. 7xxx: Remote Procedure Call errors

The set of error codes between 7000 and 7999 inclusive is reserved for errors that a specific Remote Procedure Call can return according to R.04.01.07. Which error code is used for which error condition is defined in the separate Remote Procedure Call use cases in the chapters specifying the various OCPI modules.

2.4.8. PartyRole *class*

Property	Type	Cardinality	Description
module	ModuleID	1	A module that this platform serves for this party

Property	Type	Cardinality	Description
side	InterfaceRole	1	A side (sender or receiver) this platform serves for the module and party

2.4.9. PlatformParty *class*

Property	Type	Cardinality	Description
roles	PartyRole	+	The roles that this platform serves for this party.
business_details	BusinessDetails	1	Details of this party.
party_id	PartyID	1	The Party ID of this Party.

2.4.10. PlatformParties *class*

Field Name	Type	Cardinality	Description
parties	PlatformParty	*	The Parties that the Platform sending this PlatformParties object serves to the Platform receiving this PlatformParties object
hub_party_id	PartyID	?	party ID of the Hub party of this platform

2.4.11. PointOfContact *class*

Field Name	Type	Cardinality	Description
name	UnicodeString[1..100]	1	The name of the point of contact. This can be the name of a person, but it can also be the name of a department or team like "Corporate Clients Desk".
email	AsciiString[1..320]	1	The email address of the point of contact.
telephone	AsciiString[1..15]	1	The telephone number at which the point of contact can be reached.

3. Party Issued Objects

3.1. Introduction

This section is not normative.

A lot of the functionality of OCPI has always revolved around replicating information on the state of one Party's business objects to another Party's systems. In previous versions of OCPI, there was no overarching design for how this replication was done. For each OCPI Module (CDRs, Tokens, Locations, etc), a separate description was given of how to replicate these objects. OCPI 3.0 aims to bring more structure to the way this type of information is replicated. We have made the following design decisions:

- We define the concept of a "Party Issued Object" that allows us to write a generic description of data replication, independently of the type of object being replicated. In practice this corresponds to the concept of "Client Owned Object" from previous OCPI versions, although they are defined differently.
- Party Issued Objects are transferred in HTTP requests ("push")
- Party Issued Objects are transferred at the initiative of the consumer, which in combination with the push model leads to a "pub-sub" communication pattern
- Each Party Issued Object has a version number, which allows more precise checks for being up to date than the timestamps used in previous versions
- We require updates of a single Party Issued Object to be delivered in chronological order and allow parallelization across different Party Issued Objects

3.1.1. Why such an abstract replication system?

The OCPI 3.0 pub-sub system is a significant departure from the way Client Owned Objects were exchanged in OCPI 2.2.1 and earlier. Parties with existing OCPI implementations have to invest to adapt their systems. We believe this investment is worth it because of a number of reasons, outlined in separate sections below.

3.1.1.1. A reusable system

Although implementing the pub-sub system is more complex than implementing data replication for a single OCPI module, the system can be used for *all* OCPI modules once implemented. OCPI 2.2.1 and earlier versions had little overarching description of how replication of "Client Owned Objects" was to be done.

That the pub-sub system is described separately from the functional modules should also make it easier to implement middleware products or open-source libraries that take care of managing OCPI connections and subscriptions. We hope that with OCPI 3.0, there will be more and better offerings in this space than with OCPI 2.2 and earlier.

3.1.1.2. More reliable

The pub-sub system is designed to minimize the chance that objects are not replicated by accident. In OCPI 2.2.1 this happens easily. The "push" form of replication in OCPI 2.2.1 is unreliable by design: OCPI 2.2.1 and earlier urge systems that push data to not retry failed pushes.

The reliability of data replication therefore has to come from the "pull" form of replication in OCPI 2.2.1. The "pull" form is implemented using paging and clock timestamps though. Paging is difficult to implement in a way that guarantees that the client sees all data, and easy to implement in a way that performs poorly. Clock timestamps are

well known to be unreliable in distributed systems. Therefore reliable replication with OCPI 2.2.1 is tricky in practice and impossible in theory.

OCPI 3.0 puts the burden for making sure that data consumers have a complete dataset on the data producer. Producers have to keep trying to push data to consumers until the consumer confirms to them that they received the message.

3.1.1.3. More frugal

In OCPI 2.2.1, when using the "pull" form of replication, data consumers are advised to sometimes do a "full get" of all data for a module from the producer and compare that with their own dataset. This approach is very wasteful of bandwidth and computing resources.

OCPI 3.0's pub-sub model does not require any object to ever be sent again from the producer to the consumer after the consumer has acknowledged it.

3.1.1.4. More interoperable

OCPI 2.2.1 described both a "pull" and a "push" form of data replication. Some systems only implemented one of the two. If one system implements the one and another system implements the other, they end up uninteroperable.

In OCPI 3.0, all Party Issued Object data is pushed so no implementation mismatches can arise.

3.1.1.5. More stable

With OCPI 3.0 pub-sub the data consumer tells the data producer how many concurrent push requests they can make. This helps prevent "Denial of Service" scenarios where the data producer overloads the consumer with too much data or too many requests.

Conversely, the producer can set and enforce storage limits so the producer does not have to run out of storage space when the consumer does not accept the producer's updates.

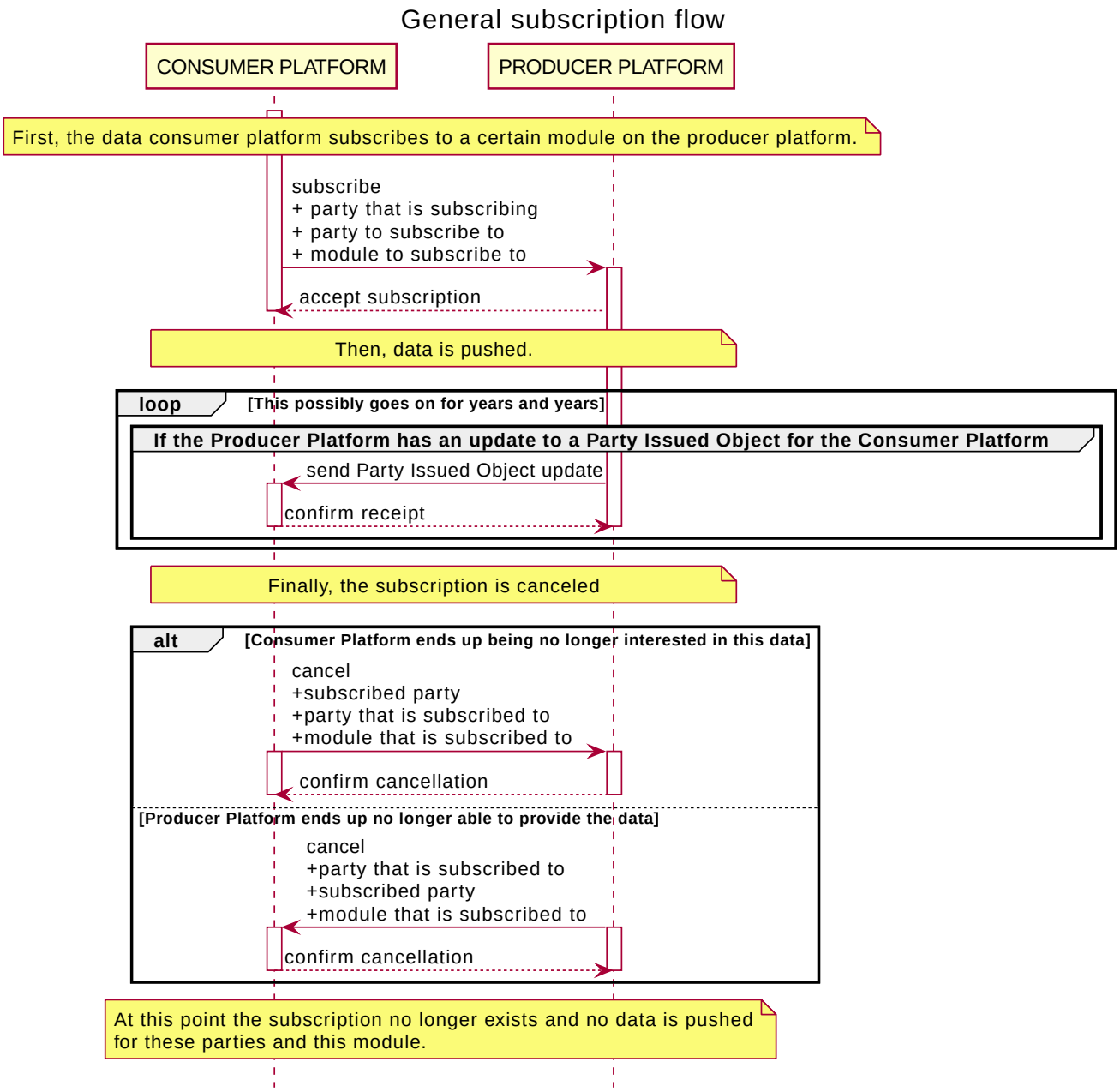
3.1.1.6. Traceability of charge authorization

The OCPI 3.0 pub-sub system assigns version numbers to updates of Party Issued Objects. The main reason for this is removing the reliance on clock times. It also brings an additional benefit however: CPOs can indicate in their Session or CDR data which version of a Token they used to authorize the Session. This can be helpful when eMSPs and CPOs are figuring out why unexpected authorizations occurred.

3.1.2. Implementing OCPI 3.0's Party Issued Object pub-sub

The OCPI 3.0 pub-sub system is centered around subscriptions. In the general flow, a data consumer subscribes to data of a certain module from a producer. After that, the producer pushes data to the consumer, until, possibly only years later, one of the two parties cancels the subscription.

This general flow is illustrated in the following sequence diagram:



3.1.2.1. How to store subscriptions

Subscriptions are durable, persistent objects. They come to be once a platform subscribes, and after that both the consumer platform and the producer platform have to keep track of them until they are canceled.

To store a subscription, you have to identify it by:

- The platform that is subscribing (i.e. the data consumer platform)
- The party that is subscribing
- The platform that is subscribed to (i.e. the data producer platform)
- The party that is subscribed to
- The module that is subscribed to (i.e. the type of objects that the consumer is asking for)

The producer platform should also store the retry interval, the maximum number of concurrent requests, and the maximum update queue size. These parameters are agreed between the platforms at the time the subscription is created.

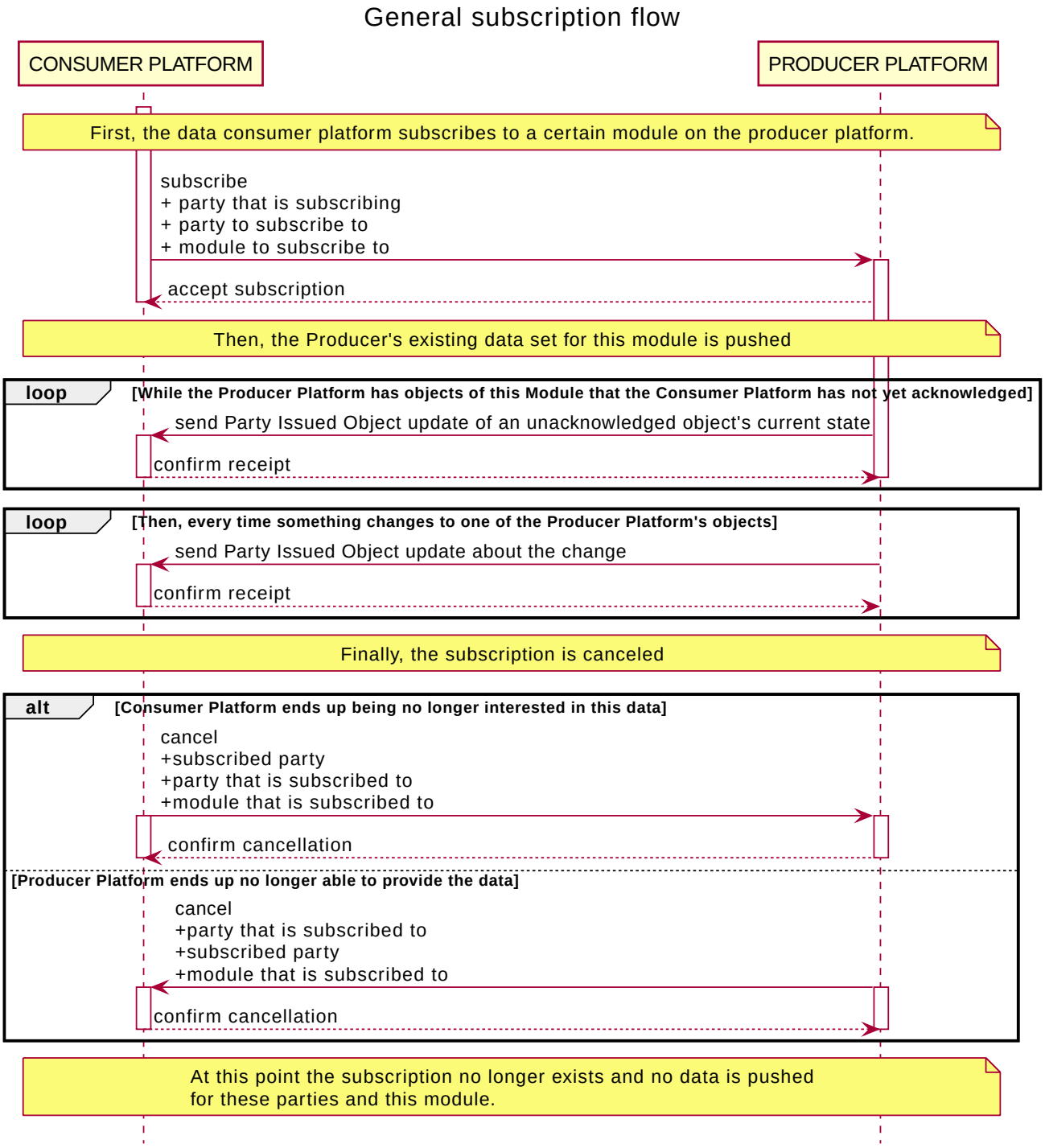
3.1.2.2. When and what to send

The diagram above states that the producer platform sends an update "If the Producer Platform has an update to a Party Issued Object for the Consumer Platform". It does not go into detail about exactly when that is the case. In fact the specification text below also does not provide more details. What a producer wants to share with a consumer, and when this is shared, is ultimately the producer's business decision to make.

There is a typical behavior that we foresee though. Typically the producer has a data set that they want to share with the consumer, and they have this set ready already when the subscription is created. For example, once an eMSP subscribes to a CPO's Locations module, the CPO will typically already have a lot of Location data that they want to share with the eMSP.

Therefore we expect that normally, immediately after the subscription is established, the producer will push the current state of all their objects to consumer. After that, they will push updates to the consumer each time something changes to one of these objects. For example, a CPO would push an update to the eMSP once a new EVSE is installed in the Location.

Taking this expectation into account, the diagram comes to look like this:



This pattern applies for those modules where objects created in the past are typically still relevant, like Locations, Tokens and Tariffs. It is less likely to apply for modules like Sessions or CDRs where consumers are usually only interested in current events, with only limited backfill of historic data if any.

3.1.2.3. Data storage on the producer side

It is important for implementers to realize that both the subscriptions and the Party Issued Objects themselves can be long-living objects that remain relevant for years. It is therefore essential that OCPI 3.0 implementations fulfill the requirements of the use cases in this chapter regardless of any redeployments, software upgrades, hardware failures and network outages that may occur while the system is running.

Some of these requirements say that the producer retry an update request until it is confirmed by the consumer. This means that producers have to store persistent queues of updates so that updates will be retried even after the

producer system has been rebooted, its network address has changed, or some other disruption of service has occurred.

We expect that producers will use message queueing systems like RabbitMQ or Kafka to implement such a persistent queue. Typically they will create a queue per subscription in such a queuing system. Producers may also want to create an "exchange" or "topic" per subscription to be able to control precisely what objects are shared at what time with which subscriptions. Some message queuing systems may already allow for such control through features like RabbitMQ's "routing keys", so that only one topic or exchange would have to exist per module.

3.1.2.4. Data storage on the consumer side

In order to process updates efficiently, a consumer will probably want to store the Party Issued Objects it receives in a database indexed by the the consumer party ID, the producer platform ID, the producer party ID and the module ID. This allows it to quickly check if it already has the Party Issued Object that it is receiving an update for, and if so, what the version of that object it has.

It is probably wise to store the Party Issued Objects as received via OCPI in a separate dataset from the live objects used in a party's business operations. This allows the consumer platform to receive the same object from multiple producer platforms, and choose which source platform to use for its business operations. It also allows the consumer party to receive data via OCPI and review it before it is taken to be the live data that affects customers.

Note also that as per requirement R.03.02.21, the consumer must accept data from the producer that may be inconsistent for business purposes. This means that a consumer may for example have to store Tariff Associations that reference Locations by ID that the consumer did not receive yet. This is another good reason for consumer implementations to separate data as received via OCPI (where consistency for business purposes cannot be enforced) from a live dataset for business operations (where consistency for business purposes can be enforced).

3.1.2.5. Optimizing throughput with many HTTP requests

OCPI 3.0 subscriptions are designed to be used with modern versions of HTTP and TLS that allow the reuse of TCP connections and TLS sessions for multiple HTTP requests. OCPI 3.0 uses an HTTP request-response exchange for every update to a Party Issued Object. Using the HTTP 1.0 connection management pattern where a new TCP connection is set up for every HTTP request would thus require TCP and TLS handshakes for every Party Issued Object update. This in turn would be a major overhead for a large CPO who has to send many updates about their charging stations and charge sessions.

We believe that at the time of writing, there is a good choice of HTTP implementations supporting connection reuse in all major programming languages. Nevertheless, some HTTP implementations may still choose the inefficient approach because of legacy or simplicity. We recommend that platform implementers who care about performance pay attention to this.

3.2. Use Cases

3.2.1. UC: 03.01 - Subscribe to Party Issued Objects of a certain Module of a certain Party

1	**Objective(s)**	**1.** Party X on Platform A starts receiving notifications about Party Issued Objects of a certain Module from Party Y on Platform B

2	**Description**	Platfrom A sends a subscription request to Platform B, stating which Party wants to subscribe and which Party and Module they want to subscribe to. Platform B answers to inform Party X and Platform A whether the subscription succeeded.
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, stating the Party wishing to subscribe, and the Party and Module that they wish to subscribe to. 2. Platform B responds, indicating if a subscription matching the request was created or not.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform A serves Party X to Party Y Platform B serves Party Y to Party X
6	**Postconditions**	Party X on Platform A is subscribed to Party Y on Platform B for the Module given in the request
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

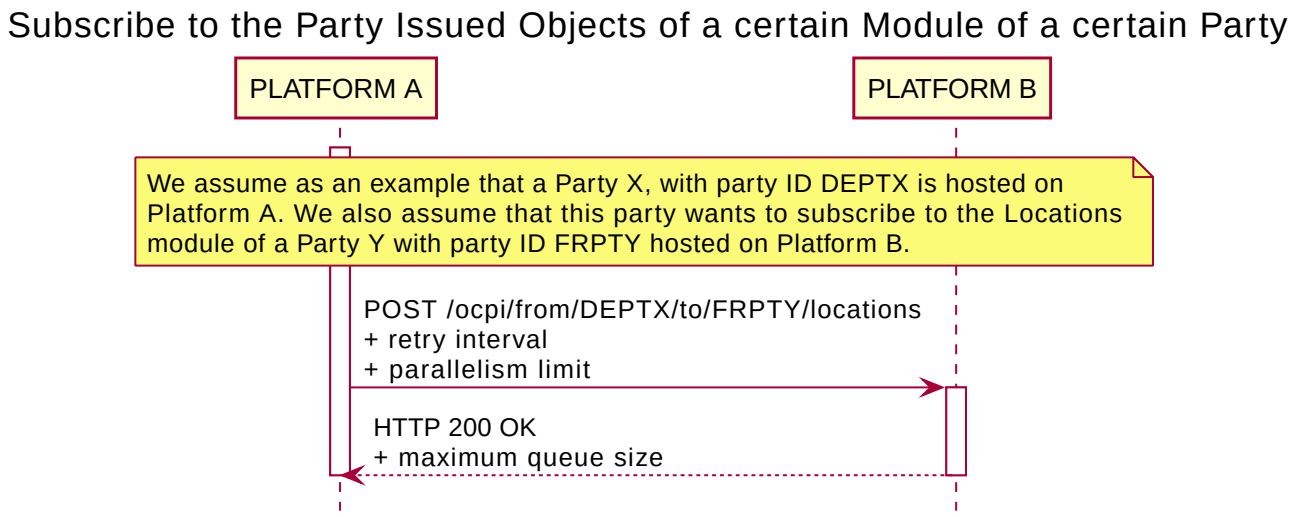


Figure 13. Sequence Diagram: Subscribe to Party Issued Objects of a certain Module of a certain Party

Table 10. UC: 03.01 Requirements

ID	Precondition	Requirement
R.03.01.01		Platform A SHALL make the request to subscribe following Make a request on behalf of a Party to a Party on another Platform.
R.03.01.02		Platform A SHALL obtain the URL to make the GET request to by appending a path segment separator and the ModuleID value for the Module to subscribe to as a relative path as described in the Make a request on behalf of a Party to a Party on another Platform.

ID	Precondition	Requirement
R.03.01.03		Platform A SHALL use POST as the HTTP request verb
R.03.01.04		Platform A SHALL put a SubscriptionRequest object in the body of its request
R.03.01.05	Platform B establishes the subscription	Platform B SHALL send a response with the status_code field set to 1000 and a SubscriptionResponse object in the payload field
R.03.01.06	Platform B processes the request but does not establish the subscription	Platform B SHOULD put an OCPI error code from Subscription error codes in the status_code field of its response

NOTE

While the parallelism_limit field in the SubscriptionRequest type *allows* Platforms to make requests in parallel, it does not require any platform to use parallel processing on a subscription at any time. Receiver platforms do not have to specify a parallelism_limit greater than 1 and sender platforms are not required to use parallel requests when they are allowed by the receiver.

NOTE

While the max_queue_size field in the SubscriptionResponse type *allows* the sender Platform to cancel a subscription, it does not absolutely require the sender to never let the update queue for a subscription grow beyond the value of this field. A sender Platform may want to temporarily allow the queue to grow bigger, for example when initializing a new subscription to a large dataset. The purpose of the field is to make resource constraints on the sender side visible on the receiver side, not to constrain the sender in how they use their resources.

3.2.2. UC: 03.02 - Send a full update of a Party Issued Object to a Subscribed Platform

1	**Objective(s)**	**1.** Party X Platform A gets an up-to-date copy of a Platform Issued Object that it is subscribed to from a Party Y on Platform B
2	**Description**	Platform B sends a message to Platform A with the Platform Issued Object data and a reference to the subscription under which the data is transferred.
3	**Actors**	Any
4	**Flow**	1. Platform B makes a request to Platform A, carrying a Party Issued Object's identifier, version and representation, and referencing the subscription under which the data is transferred.
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	Party X on Platform A has updated information on a Party Issued Object from Party Y on Platform B
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.

8	**Remark(s)**	Where we use "Platform A's copy of the Party Issued Object" in the requirements below, we mean the copy of the Party Issued Object that Platform A has obtained through previous request-response exchanges, as identified by the subscription identity, the issuer party ID and the object ID

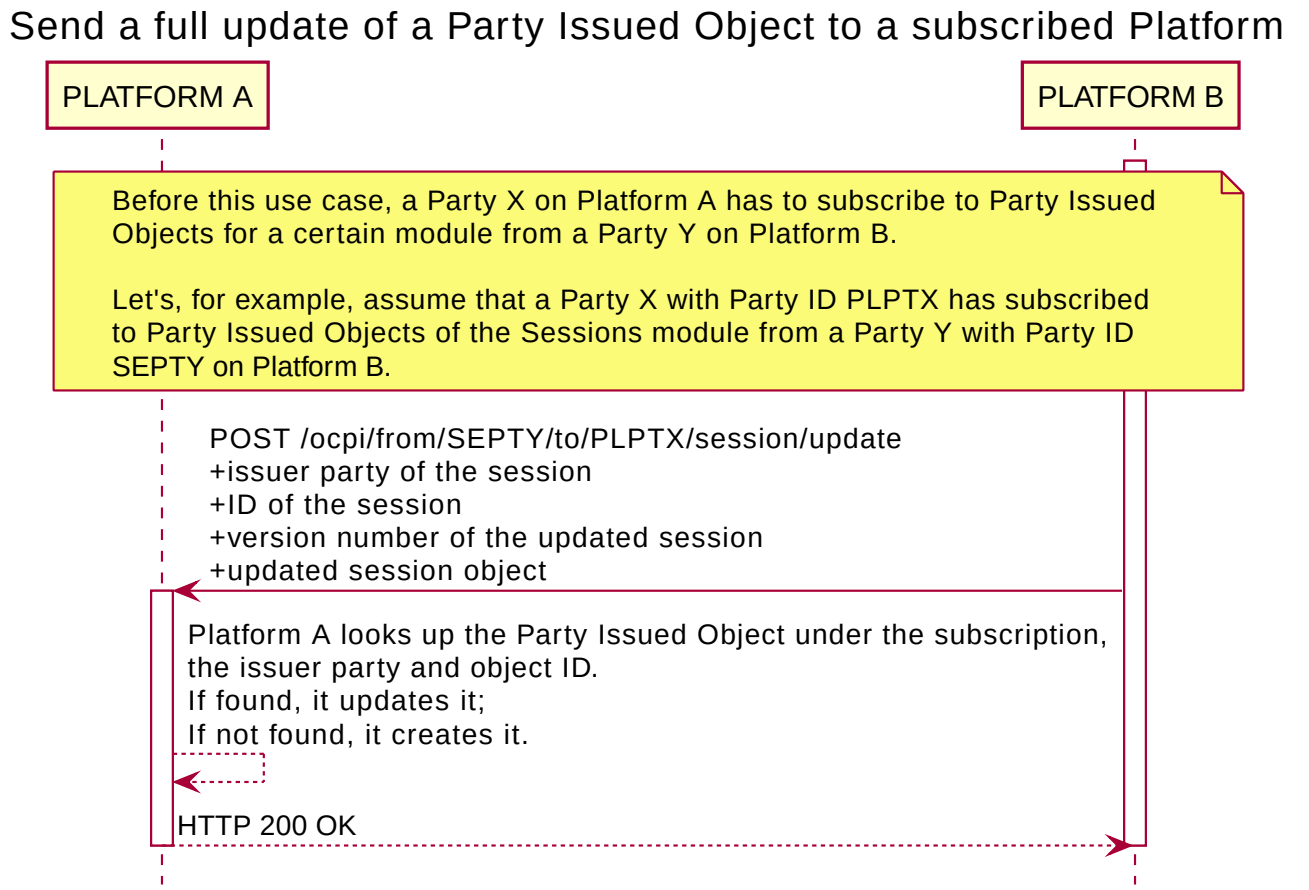


Figure 14. Sequence Diagram: Send a full update of a Party Issued Object to a Subscribed Platform

Table 11. UC: 03.02 Requirements

ID	Precondition	Requirement
R.03.02.01		Platform B SHALL make the request to Platform A following Make a request on behalf of a Party to a Party on another Platform.
R.03.02.02		Platform B SHALL make the request to Platform A with the POST request verb
R.03.02.03		Platform B SHALL set the sender Party ID in the URL path of its request to Platform A to the Party ID of the Party that was subscribed to according to Subscribe to Party Issued Objects of a certain Module of a certain Party
R.03.02.04		Platform B SHALL set the receiver Party ID in the URL path of its request to Platform A to the Party ID of the Party that subscribed according to Subscribe to Party Issued Objects of a certain Module of a certain Party

ID	Precondition	Requirement
R.03.02.05		Platform B SHALL add the ModuleID of the Module that was subscribed to according to Subscribe to Party Issued Objects of a certain Module of a certain Party as the first custom path segment
R.03.02.06		Platform B SHALL add "update" as the second custom path segment of the URL path in its request to Platform A
R.03.02.07		Platform B SHALL send a PartyIssuedObjectUpdate object in the request body of the request
R.03.02.08	The preconditions of R.03.02.12 up to and including R.03.02.15 do not apply	Platform A SHALL update or create its copy of the Party Issued Object with the issuer party and ID and Module from the request body to be the same as the object in the "payload" field of the request body
R.03.02.09	Platform A does not have a copy of the Party Issued Object yet	Platform A SHALL create its copy of the Party Issued Object with the issuer party and ID and Module of the request body to be the same as the object in the "payload" field of the request body
R.03.02.10	Platform A successfully stored a version of its copy of the Party Issued object according to R.03.02.08 or R.03.02.09	Platform A SHALL store that the version of its copy of the Party Issued Object is that version that is given in the "version" field of the PartyIssuedObjectUpdate object in the request body
R.03.02.11	Platform A successfully updated its version of the Party Issued Object according to R.03.02.10	Platform A SHALL send a response to Platform B with a OcpiResponse object in the response body. The OcpiResponse object SHOULD NOT have a payload and the value of the "status_code" field SHALL be 1000.
R.03.02.12	The value of the "version" field in the PartyIssuedObjectUpdate object in the request body is smaller than or equal to the version of Platform A's copy of the Party Issued Object	Platform A SHALL NOT modify its copy of the Party Issued Object
R.03.02.13	Party X on Platform A is not subscribed to Party Y on Platform B for the Module given in the URL path of the request	Platform A SHALL send a response to Platform B with the "status_code" field in the OcpiResponse object set to 6002.
R.03.02.14	The value of the "version" field in the PartyIssuedObjectUpdate object in the request body is smaller than or equal to the version of Platform A's copy of the Party Issued Object and the precondition of R.03.02.11 does not apply	Platform A SHALL send a response to Platform B with a OcpiResponse object in the response body. The OcpiResponse object SHOULD NOT have a payload and the value of the "status_code" field SHALL be 1000.
R.03.02.15	The value of the "payload" field in the PartyIssuedObjectUpdate object in the request body of the request does not conform to the schema for the Module and the preconditions of R.03.02.11 and R.03.02.12 do not apply	Platform A SHALL send a response to Platform B with the "status_code" field in the OcpiResponse object set to 6001

ID	Precondition	Requirement
R.03.02.16	Platform B does not get a response that has a "status_code" field with a value of 1000, 6001 or 6002 from Platform A within the timeout period that it received from Platform A according to Handshake OCPI Connection Parameters	Platform B SHALL retry the update according to Retry an update of a Party Issued Object to a Subscribed Platform
R.03.02.17		Platform B SHALL NOT make its POST request to Platform A while the number of POST requests in progress made in the context of use case 11 is equal to or greater than the parallelism limit set for this subscription by Platform A during use case 10. A request is considered "in progress" for this requirement if and only if it has not been responded to by Platform A and Platform A's request timeout as given during use case 3 has not elapsed since the request was sent.
R.03.02.18		Platform B SHALL NOT make its POST request to Platform A while another POST request is in progress made in the context of use case 11, for the same subscription, and with the same Party Issued Object Party ID and Party Issued Object ID. A request is considered "in progress" for this requirement if and only if it has not been responded to by Platform A and Platform A's request timeout as given during use case 3 has not elapsed since the request was sent.
R.03.02.19		Platform A SHALL NOT use the data from the request for any Party other than the subscribed Party, except when following use cases 19 or 21 given in this chapter.
R.03.02.20		Platform B SHALL NOT set the "id" field of the PartyIssuedObjectUpdate request in the request body to a string with more than 36 code points, unless it is making the request to issue a credit CDR according to Send a Credit CDR.
R.03.02.21	The payload in the PartyIssuedObjectUPdate object in the request contains data that is inconsistent at the business level with other data that Platform A holds for Party X, like when the payload object contains dangling references to other business objects from other modules that may not yet have been transferred.	Platform A SHALL NOT report this inconsistency to Platform B in the response of this use case.

NOTE

Requirement R.03.02.21 should not be taken to mean that Platform A and Party X should not do business-level validation at all. The point is that Platform A or Party X should not make Platform B's update request fail because of such validations. For Party Issued Objects, the sender is the source of truth. The consumer has to store what the sender sends, even if what the sender sends is nonsense

in the receiver's eyes. For dangling references, like Tariff Associations that reference Locations or Tariffs that don't exist in the receiver's system, the receiver should accept them and try to follow the references only once they have to be used in business operations. For other types of inconsistencies, the receiver may contact the sender and seek to receive them using communication methods other than OCPI.

3.2.3. UC: 03.03 - Retry an update of a Party Issued Object to a Subscribed Platform

1	**Objective(s)**	**1.** Party X Platform A gets an up-to-date copy of a Platform Issued Object that it is subscribed to from a Party Y on Platform B
2	**Description**	Updates to receiver's copies of Party Issued Objects, as performed according to use case Send a full update of a Party Issued Object to a Subscribed Platform, may fail for various reasons. Reasons for failures include, but are certainly not limited to, network disruptions, infrastructure malfunction, planned maintenance and software bugs. This use case serves to make OCPI 3.0 connections recover from such failures without intervention by human operators. It does so by making the sender retry the update until either the update succeeds or the whole subscription is terminated due to the sender's persistent queue growing beyond its size limit.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform B makes a request to Platform A, carrying a Party Issued Object's identifier, version and representation, and referencing the subscription under which the data is transferred.
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	Party X on Platform A has updated information on a Party Issued Object from Party Y on Platform B
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

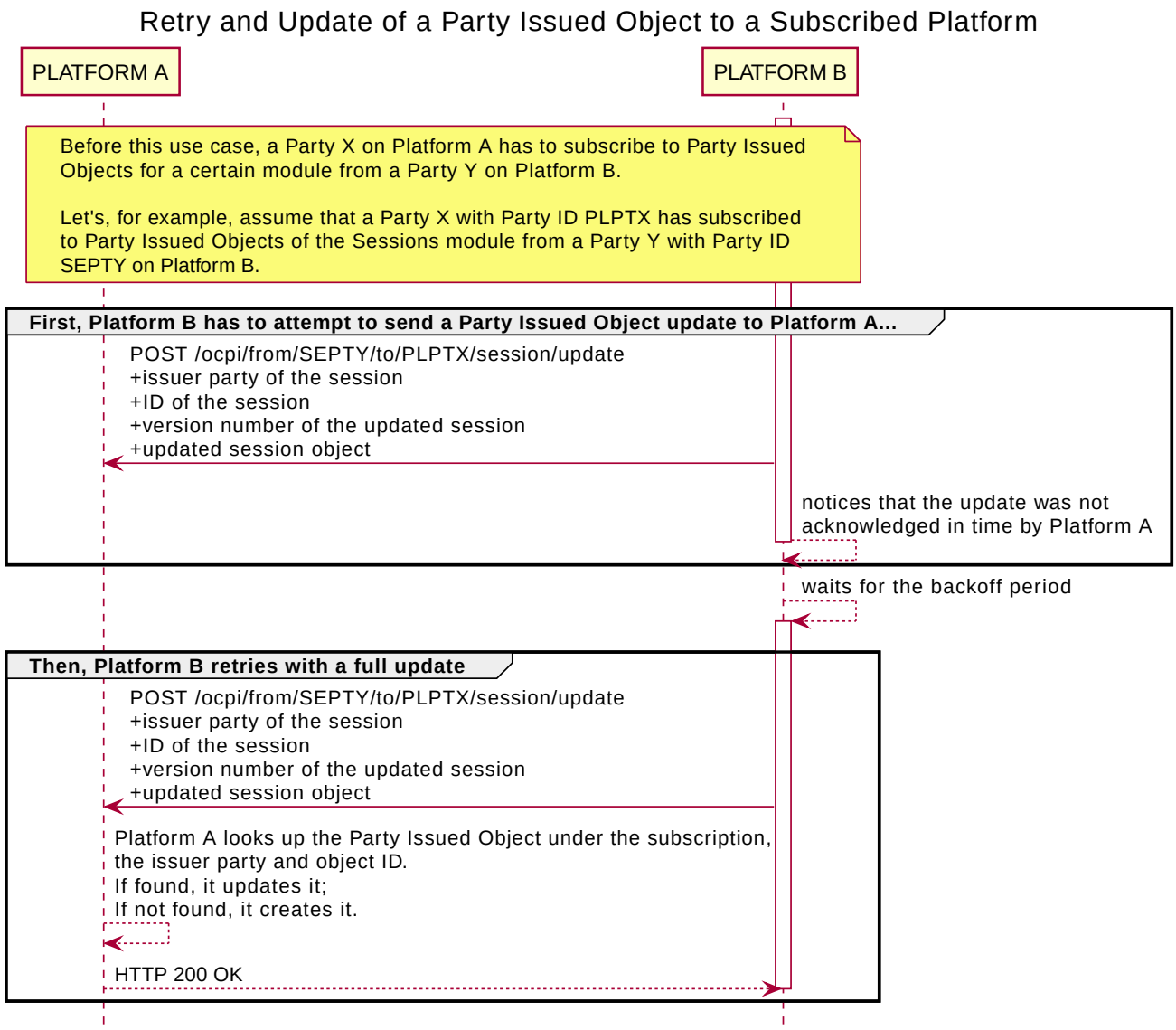


Figure 15. Sequence Diagram: Retry an update of a Party Issued Object to a Subscribed Platform

Table 12. UC: 03.03 Requirements

ID	Precondition	Requirement
R.03.03.01		Platform B SHALL NOT make an update request according to Send a Full Update of a Party Issued Object to a Subscribed Platform for the same subscription, Party Issued Object Party ID, Party Issued Object ID and Party Issued Object version until the backoff period has elapsed. The backoff period is *2 ^ (n - 1) * p* seconds, where *n* is the number of times an update request for this party issued object at this version was made without getting a success response, and *p* is the subscription's retry interval as Platform A gave it to Platform B by during use case Subscribe to Party Issued Objects of a certain Module of a certain Party.

ID	Precondition	Requirement
R.03.03.02		Platform B SHOULD do a full update of the same subscription, Party Issued Object Party ID, Party Issued Object ID and Party Issued Object version according to Send a Full Update of a Party issued Object to a Subscribed Platform as immediately as possible after the backoff period has elapsed
R.03.03.03		Platform B MAY discard the update that it is holding for the backoff period according to R.03.03.01 after it did a full update in the same subscription with the same Party Issued Object Party ID and Party Issued Object ID and a Party issued Object version that is greater than the Party Issued Object version in the update that it is holding.

3.2.4. UC: 03.04 - Cancel a Subscription as the Platform receiving data

1	**Objective(s)**	**1.** The subscription is terminated and the data receiver Platform no longer receives data belonging to this subscription
2	**Description**	Once a subscriber Platform loses interest in data they are subscribed to, they will want to end the subscription and stop receiving data. They can do so by sending a request to cancel the subscription to the data sender Platform.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, carrying the parameters identifying the subscription and the desire to cancel in the URL 2. Platform B stops sending data to Platform A and acknowledges the cancellation in a response to Platform A
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	Party X on Platform A is not subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

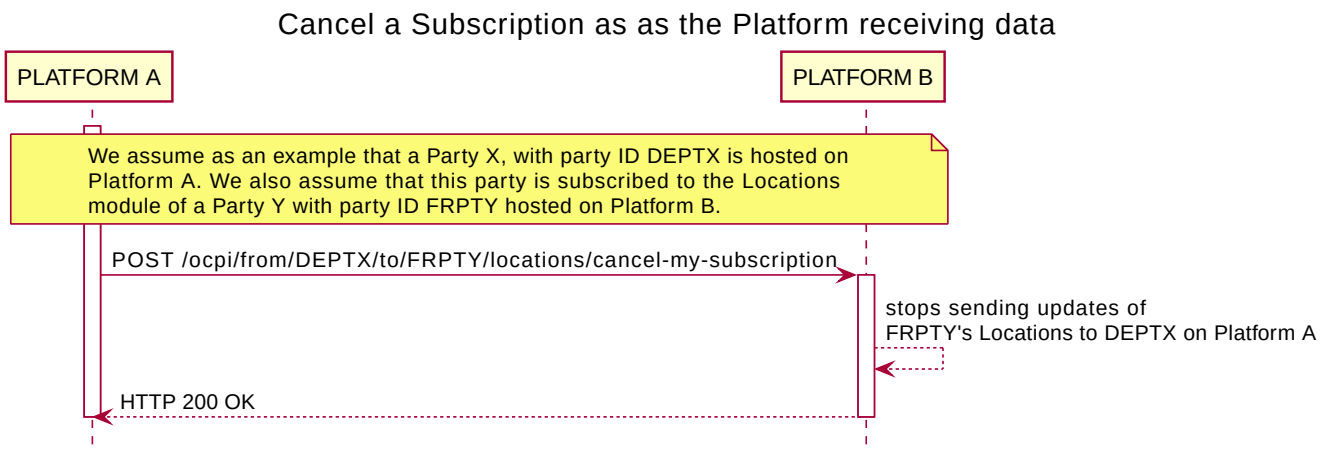


Figure 16. Sequence Diagram: Cancel a Subscription as the Platform receiving data

Table 13. UC: 03.04 Requirements

ID	Precondition	Requirement
R.03.04.01		Platform A SHALL make the request to Platform B following Make a request on behalf of a Party to a Party on another Platform.
R.03.04.02		Platform A SHALL obtain the URL to make the GET request to by appending as path segments the ModuleID value for the subscription to cancel and the string "cancel-my-subscription" to the path as described in the Make a request on behalf of a Party to a Party on another Platform.
R.03.04.03		Platform A SHALL use POST as the HTTP request verb
R.03.04.04		Platform A SHALL NOT put a request body in its request
R.03.04.05	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL	Platform B SHALL NOT begin with use case 12 for that subscription after sending its response to Platform A's cancellation request
R.03.04.06	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL	Platform B SHOULD discard updates that are waiting for a backoff period according to Retry an update of a Party Issued Object to a Subscribed Platform
R.03.04.06	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL	Platform B SHALL send a response with the status_code field set to 1000
R.03.04.08	Platform B cannot find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL	Platform B SHALL send a response with the status_code field set to 5005

3.2.5. UC: 03.05 - Cancel a Subscription as the Platform sending data

1	**Objective(s)**	**1.** The subscription is terminated and the data receiver Platform no longer receives data belonging to this subscription
2	**Description**	Once a data sender Platform is no longer able to deliver the data that another Platform subscribed to them for, they will want to end the subscription so that the data receiver Platform knows they will no longer receive updates and their copy of the data is becoming stale. The data sender Platform can do this by sending a request to cancel the subscription to the data receiver Platform.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform B stops sending data updates to Platform A in the context of the subscription 2. Platform B makes a request to Platform A, carrying the parameters identifying the subscription and the desire to cancel in the URL
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	Party X on Platform A is not subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

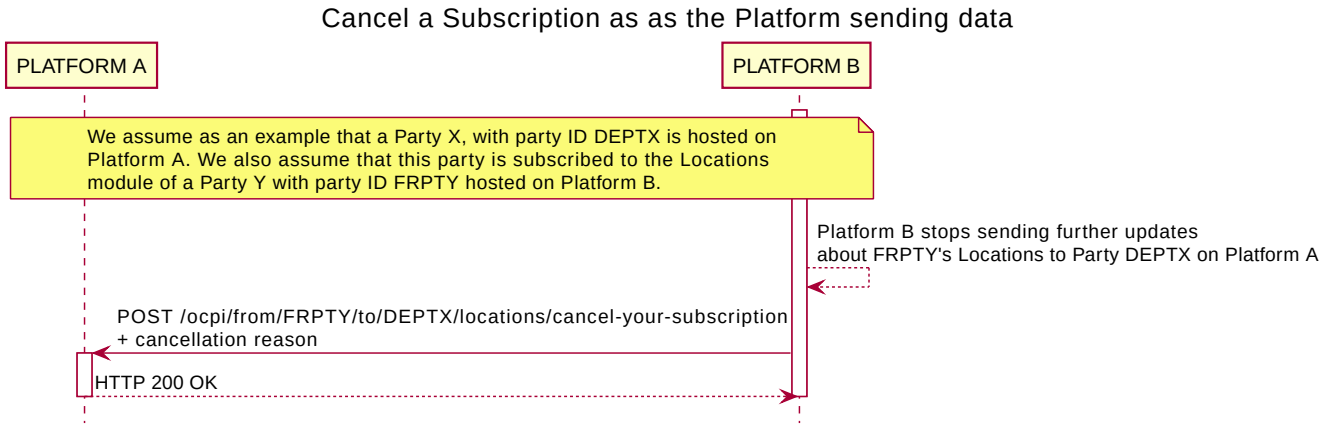


Figure 17. Sequence Diagram: Cancel a Subscription as the Platform sending data

Table 14. UC: 03.05 Requirements

ID	Precondition	Requirement
R.03.05.01		Platform B SHALL make the request to Platform A following Make a request on behalf of a Party to a Party on another Platform.

ID	Precondition	Requirement
R.03.05.02		Platform B SHALL obtain the URL to make the GET request to by appending as path segments the ModuleID value for the subscription to cancel and the string "cancel-your-subscription" to the path as described in the Make a request on behalf of a Party to a Party on another Platform.
R.03.05.03		Platform B SHALL use POST as the HTTP request verb
R.03.05.04		Platform B SHALL put a SubscriptionCancellation object in the body of its request
R.03.05.05	Platform A can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL	Platform A SHALL send a response with the status_code field set to 1000
R.03.05.08	Platform A cannot find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL	Platform A SHALL send a response with the status_code field set to 5005
R.03.05.09		Platform B SHALL NOT begin with use case 12 for the subscription being cancelled after sending its cancellation request to Platform A

3.2.6. UC: 03.06 - Check subscription state on sender side as the platform receiving data

1	**Objective(s)**	**1.** The receiving platform gets to know how many updates the sending platform is queuing for them
2	**Description**	A receiving platform may want to check that they are not behind on processing updates and that they are not using a lot of the sending platform's storage space. Therefore they can request the state of the subscription at the sender.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform A requests the state of a certain subscription from Platform B 2. Platform B responds to Platform A, with the subscription state in the response
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	None in particular
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.

8	**Remark(s)**	The data in the response from Platform B may be outdated by the time Platform A receives them.

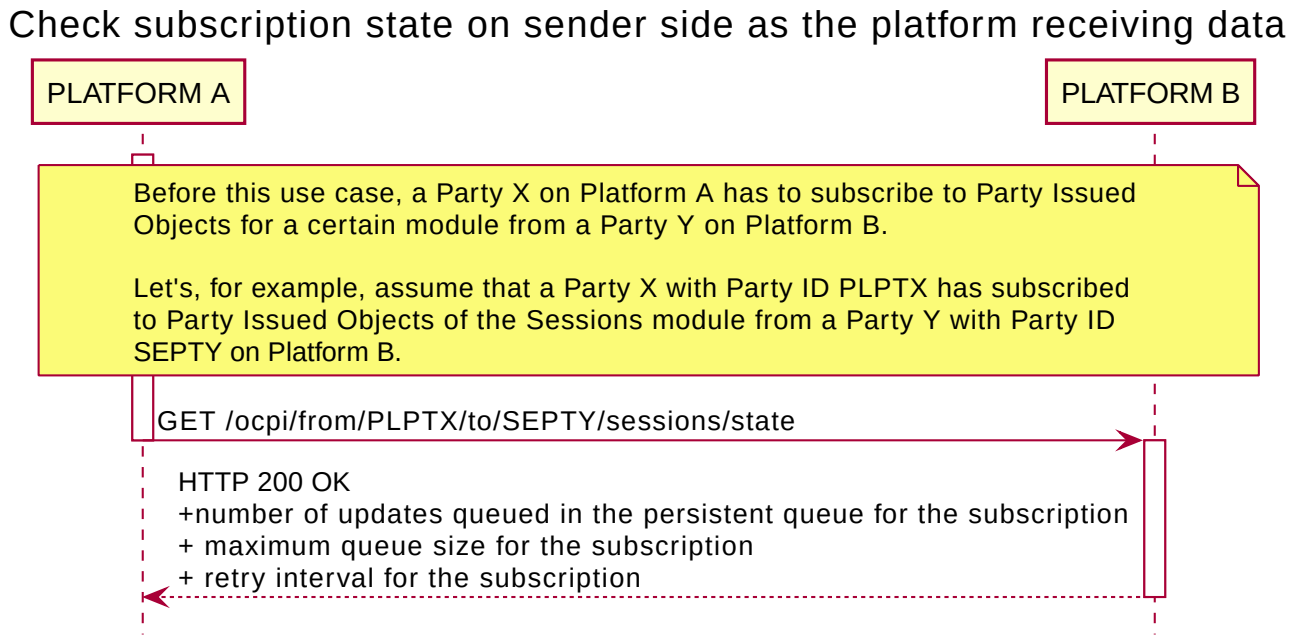


Figure 18. Sequence Diagram: Check subscription state on sender side as the platform receiving data

Table 15. UC: 03.06 Requirements

ID	Precondition	Requirement
R.03.06.01		Platform A SHALL make the request for the subscription state following Make a request on behalf of a Party to a Party on another Platform.
R.03.06.02		Platform A SHALL obtain the URL to make the GET request to by appending to the URL path as described in Make a request on behalf of a Party to a Party on another Platform: a path segment separator, the ModuleID value for the Module of the subscription, another path segment separator, and the string "state" as a last segment.
R.03.06.03		Platform A SHALL use GET as the HTTP request verb.
R.03.06.04	Platform B indeed has a subscription where it is the subscription's sender platform, the receiver party of the GET request URL is the subscription's sender party, Platform A is the subscription's receiver platform and the sender party of the GET request URL is the subscription's receiver party	Platform B SHALL send a response with the status_code field set to 1000 and a SubscriptionState object in the payload field.

ID	Precondition	Requirement
R.03.06.05	Platform B does not have a subscription where it is the subscription's sender platform, the receiver party of the GET request URL is the subscription's sender party, Platform A is the subscription's receiver platform and the sender party of the GET request URL is the subscription's receiver party	Platform B SHALL send a response with the status_code field set to 5005.

3.2.7. UC: 03.07 - Renegotiate Subscription Parameters as the Platform receiving data

1	**Objective(s)**	A platform subscribed to another platform gets the sender platform to agree to a new set of parameters for a subscription.
2	**Description**	Sometimes, due to changes in data volumes or business requirements or available computing resources, platforms will want to change the parameters for a subscription. This use case lets the receiving platform in the subscription take the initiative to do so.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, carrying the parameters identifying the subscription and the desire to renegotiate parameters in the URL, and carrying the proposed new parameters in the request body. 2. Platform B responds, indicating in the response body if it agrees to the new parameters or not.
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	One of: **1.** Platform A and and Platform B agreed new parameters for a subscription between them; or **2.** Platform A knows that Platform B did not agree to Platform A's proposed new subscription parameters.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

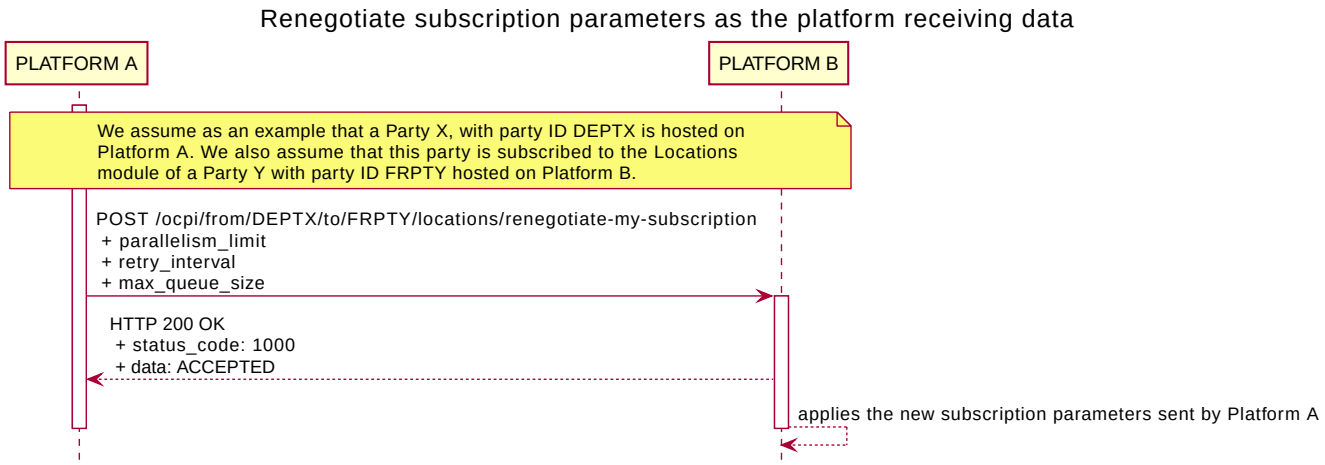


Figure 19. Sequence Diagram: Renegotiate Subscription Parameters as the Platform receiving data

Table 16. UC: 03.07 Requirements

ID	Precondition	Requirement
R.03.07.01		Platform A SHALL make the request to Platform B following Make a request on behalf of a Party to a Party on another Platform.
R.03.07.02		Platform A SHALL obtain the URL to make the GET request to by appending as path segments the ModuleID value for the subscription to cancel and the string "renegotiate-my-subscription" to the path as described in the Make a request on behalf of a Party to a Party on another Platform.
R.03.07.03		Platform A SHALL use POST as the HTTP request verb
R.03.07.04		Platform A SHALL put a SubscriptionParameterProposal object in the request body in its request
R.03.07.05	Platform B cannot find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL.	Platform B SHALL send a response with the status_code field set to 5005.
R.03.07.06	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is willing and able to change the subscription parameters as proposed in the request by Platform A.	Platform B SHALL send a response with the status_code field set to 1000 and the data field set to ACCEPTED.
R.03.07.07	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is willing and able to change the subscription parameters as proposed in the request by Platform A.	Platform B SHALL start applying the new subscription parameters for new instances of the Send a Full Update of a Party Issued Object to a Subscribed Platform for this subscription.

ID	Precondition	Requirement
R.03.07.08	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is not willing or able to change the subscription parameters as proposed in the request by Platform A.	Platform B SHALL send a response with the status_code field set to 1000 and the data field set to REJECTED.
R.03.07.09	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is not willing or able to change the subscription parameters as proposed in the request by Platform A.	Platform B SHALL keep applying the previously agreed subscription parameters for new instances of the Send a Full Update of a Party Issued Object to a Subscribed Platform for this subscription.

3.2.8. UC: 03.08 - Renegotiate Subscription Parameters as the Platform sending data

1	**Objective(s)**	A platform that another platform subscribed to gets the receiver platform to agree to a new set of parameters for a subscription.
2	**Description**	Sometimes, due to changes in data volumes or business requirements or available computing resources, platforms will want to change the parameters for a subscription. This use case lets the sending platform in the subscription take the initiative to do so.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform B makes a request to Platform A, carrying the parameters identifying the subscription and the desire to renegotiate parameters in the URL, and carrying the proposed new parameters in the request body. 2. Platform A responds, indicating in the response body if it agrees to the new parameters or not.
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	One of: **1.** Platform A and and Platform B agreed new parameters for a subscription between them; or **2.** Platform B knows that Platform A did not agree to Platform B's proposed new subscription parameters.
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

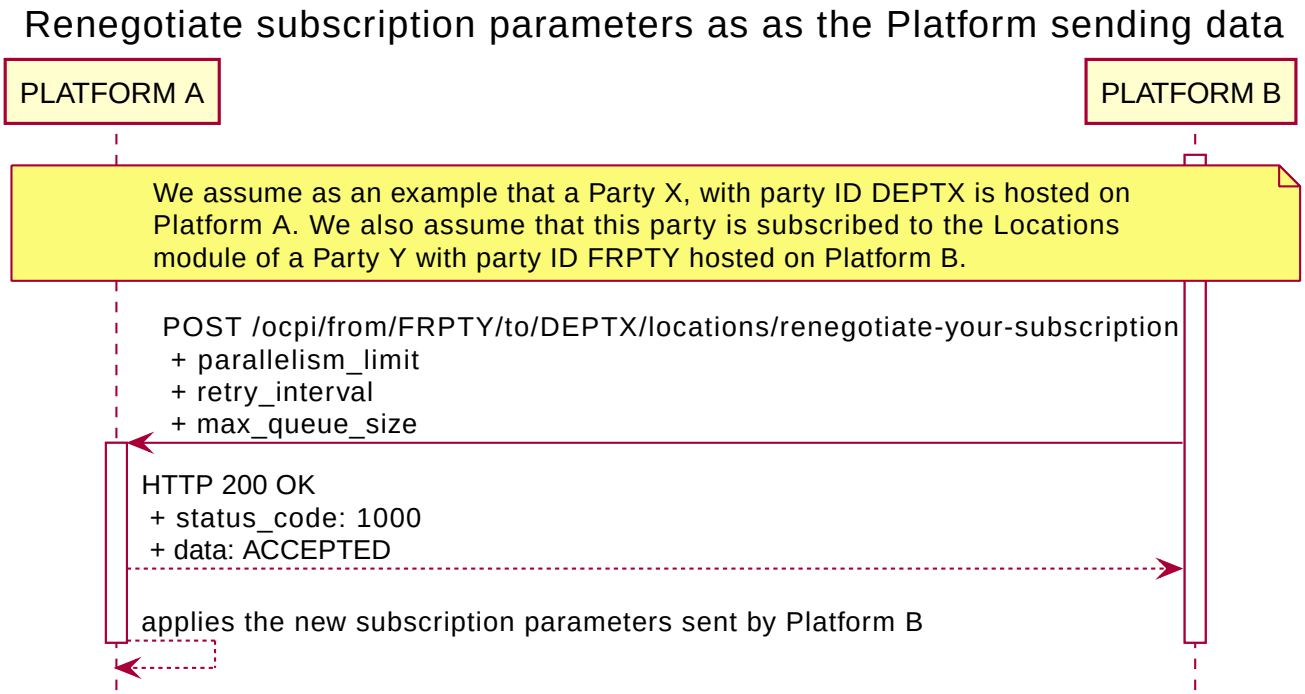


Figure 20. Sequence Diagram: Renegotiate Subscription Parameters as the Platform sending data

Table 17. UC: 03.08 Requirements

ID	Precondition	Requirement
R.03.08.01		Platform B SHALL make the request to Platform A following Make a request on behalf of a Party to a Party on another Platform.
R.03.08.02		Platform B SHALL obtain the URL to make the GET request to by appending as path segments the ModuleID value for the subscription to cancel and the string "renegotiate-your-subscription" to the path as described in the Make a request on behalf of a Party to a Party on another Platform.
R.03.08.03		Platform B SHALL use POST as the HTTP request verb
R.03.08.04		Platform B SHALL put a SubscriptionParameterProposal object in the request body in its request
R.03.08.05	Platform A cannot find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL.	Platform A SHALL send a response with the status_code field set to 5005.
R.03.08.06	Platform A can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is willing and able to change the subscription parameters as proposed in the request by Platform B.	Platform A SHALL send a response with the status_code field set to 1000 and the data field set to ACCEPTED.

ID	Precondition	Requirement
R.03.08.07	Platform A can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is willing and able to change the subscription parameters as proposed in the request by Platform B.	Platform A SHALL start applying the new subscription parameters for new instances of the Send a Full Update of a Party Issued Object to a Subscribed Platform for this subscription.
R.03.08.08	Platform A can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is not willing or able to change the subscription parameters as proposed in the request by Platform B.	Platform A SHALL send a response with the status_code field set to 1000 and the data field set to REJECTED.
R.03.08.09	Platform A can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL and is not willing or able to change the subscription parameters as proposed in the request by Platform B.	Platform A SHALL keep applying the previously agreed subscription parameters for new instances of the Send a Full Update of a Party Issued Object to a Subscribed Platform for this subscription.

3.2.9. UC: 03.09 - Request immediate retry of all pending updates for a Subscription

1	**Objective(s)**	A platform subscribed to another platform gets the sender platform to immediately retry all its updates that are waiting to be retried.
2	**Description**	Sometimes Party Issued Object updates will fail to be delivered for a clearly identifiable, temporary pattern of failure, like a network outage or a broken version of software being deployed on the receiver Platform. In such case, once this temporary pattern of failure has been fixed, the receiver platform will want to process the updates to be retried as soon as possible, rather than wait for their retry periods to expire. Retry periods can become quite long after repeated failures due to the exponential backoff, and this use case gives the receiver Platform a way to quickly catch up again.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, carrying the parameters identifying the subscription and the desire to have all pending updates retried as soon as possible within the parallelism limit of the subscription and the computing resources available to Platform B. 2. Platform B retries all pending updates for the subscription.
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	None are guaranteed.

7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	Note that the request from Platform A *allows* Platform B to send the updates sooner than backoff requires, but does not oblige Platform B to do anything beyond sending a response to the request. Platform B can disregard the request, for example if it has already spent to many resources on retrying updates to Platform A, or if it just hasn't implemented the logic for cancelling the backoff wait.

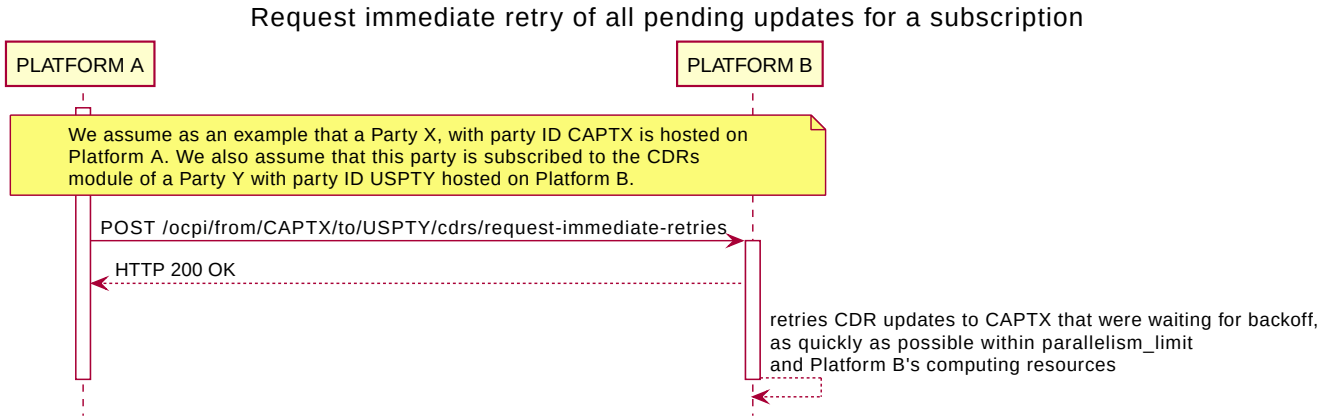


Figure 21. Sequence Diagram: Request immediate retry of all pending updates for a Subscription

Table 18. UC: 03.09 Requirements

ID	Precondition	Requirement
R.03.09.01		Platform A SHALL make the request to Platform B following Make a request on behalf of a Party to a Party on another Platform.
R.03.09.02		Platform A SHALL obtain the URL to make the GET request to by appending as path segments the ModuleID value for the subscription to request immediate retries for and the string "request-immediate-retries" to the path as described in the Make a request on behalf of a Party to a Party on another Platform.
R.03.09.03		Platform A SHALL use POST as the HTTP request verb
R.03.09.04	Platform B cannot find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL.	Platform B SHALL send a response with the status_code field set to 5005.
R.03.09.05	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL.	Platform B SHALL send a response with the status_code field set to 1000.

ID	Precondition	Requirement
R.03.09.06	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL.	Platform B MAY make update requests according to Send a Full Update of a Party Issued Object to a Subscribed Platform for combinations of subscription, Party Issued Object Party ID, Party Issued Object ID and Party Issued Object version, such that the subscription is the subscription found according to the precondition, and such that Platform B was waiting for a backoff period according to R.03.03.01. Platform B may make these requests without waiting for the rest of the backoff period after it received Platform A's request.

3.2.10. UC: 03.10 - Request full update of a certain Party Issued Object

1	**Objective(s)**	A platform subscribed to another platform gets the sender platform to send it a full update of a certain Party Issued Object.
2	**Description**	When a receiver Platform is not sure that they have the latest version of a Party Issued Object, they can request the sender Platform to send it to them according to this use case.
3	**Actors**	Any
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, carrying the parameters identifying the subscription and the Party Issued Object to send an update for. 2. Platform B sends a full update of this Party Issued Object to Platform A.
5	**Preconditions**	Party X on Platform A is subscribed to Party Issued Objects from Party Y on Platform B for a certain Module according to Subscribe to Party Issued Objects of a certain Module of a certain Party
6	**Postconditions**	One of: * Party X on Platform A is going to receive a full update of the Party Issued Object that they identified in their request from Party Y on Platform B; or * Party X on Platform A knows that Party Y on Platform B will not send them a full update, and they know why they will not get such update.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

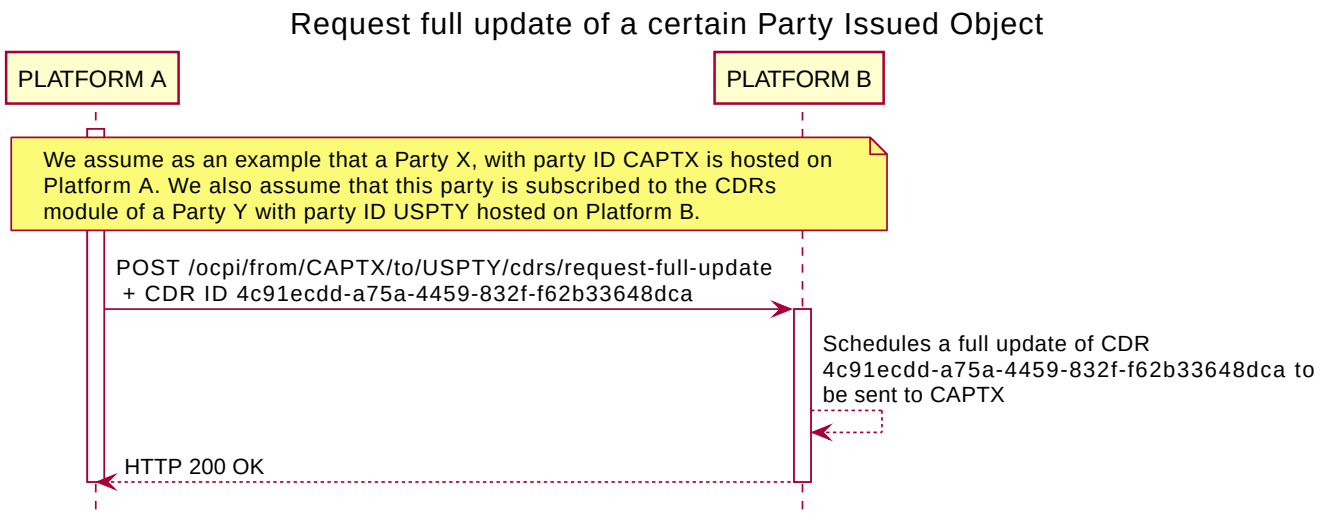


Figure 22. Sequence Diagram: Request full update of a certain Party Issued Object

Table 19. UC: 03.10 Requirements

ID	Precondition	Requirement
R.03.10.01		Platform A SHALL make the request to Platform B following Make a request on behalf of a Party to a Party on another Platform.
R.03.10.02		Platform A SHALL obtain the URL to make the GET request to by appending as path segments the ModuleID value for the subscription to cancel and the string "request-full-update" to the path as described in the Make a request on behalf of a Party to a Party on another Platform.
R.03.10.03		Platform A SHALL use POST as the HTTP request verb
R.03.10.04		Platform A SHALL put a PartyIssuedObjectReference object in the request body in its request
R.03.10.05	Platform B cannot find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL.	Platform B SHALL send a response with the status_code field set to 5005.
R.03.10.06	Platform B can find a subscription as described in the precondition of R.03.10.05 but cannot find a Party Issued Object issued within the context of this subscription with the ID given in the request body by Platform A.	Platform B SHALL send a response with the status_code field set to 5006.
R.03.10.07	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL, and also issued an object with the ID given in the request to Party X in the context of this subscription.	Platform B SHALL send a response with the status_code field set to 1000.

ID	Precondition	Requirement
R.03.10.08	Platform B can find a subscription with Platform A as the receiver platform, Platform B as the sender platform, and the parties and ModuleID given in the URL, and also issued an object with the ID given in the request to Party X in the context of this subscription.	Platform B SHALL send a Party Issued Object update for the latest version of the object with the ID given in Platform A's request, issued in the context of the subscription described in the precondition, as soon as Platform B has computing resources available to do so.

3.2.11. UC: 03.11 - Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub

1	**Objective(s)**	**1.** Platform A starts receiving notifications about Party Issued Objects of a certain Module from Party Y on Platform B. The notifications are addressed to Platform A's hub party ID. Platform A uses these notifications to provide data on Party Y's Party Issued Objects to all the parties that Platform A serves to Platform B.
2	**Description**	When a Party with a self-hosted Platform connects to a Hub, they will typically want to send their Party Issued Objects to the Hub once and have the Hub distribute them to Parties on other Platforms connected to the Hub. This use case allows for that to happen.
3	**Actors**	Roaming Hub
4	**Flow**	1. Platform A makes a request to Platform B, giving the Party and Module that they wish to subscribe to, and giving the Hub Party as the Party wishing to subscribe 2. Platform B responds, indicating if a subscription matching the request was created or not.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform A offers Hub functionality to Platform B Platform B serves Party Y to Platform A
6	**Postconditions**	Platform A's Hub is subscribed to Party Y on Platform B for the Module given in the request
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	In a typical scenario, Platform A would be a platform operated by a Roaming Hub and Platform B would be platform operated by a CPO, eMSP or similar self-hosting market party. This use case corresponds to Broadcast Push in OCPI 2.2.1 and 2.2, although the technical implementation is quite different because the initiative for the exchange has moved from the data producer to the data consumer. Typically, Hubs will give a self-hosting party connecting to them the ability to configure, in a web portal or such, which Parties the Hub's Platform serves to the connected self-hosted Platform. So indirectly the self-hosting company (Platform B) still decides who has access to their data, although they have to trust the Roaming Hub (the operator of Platform A) to enforce their decisions.

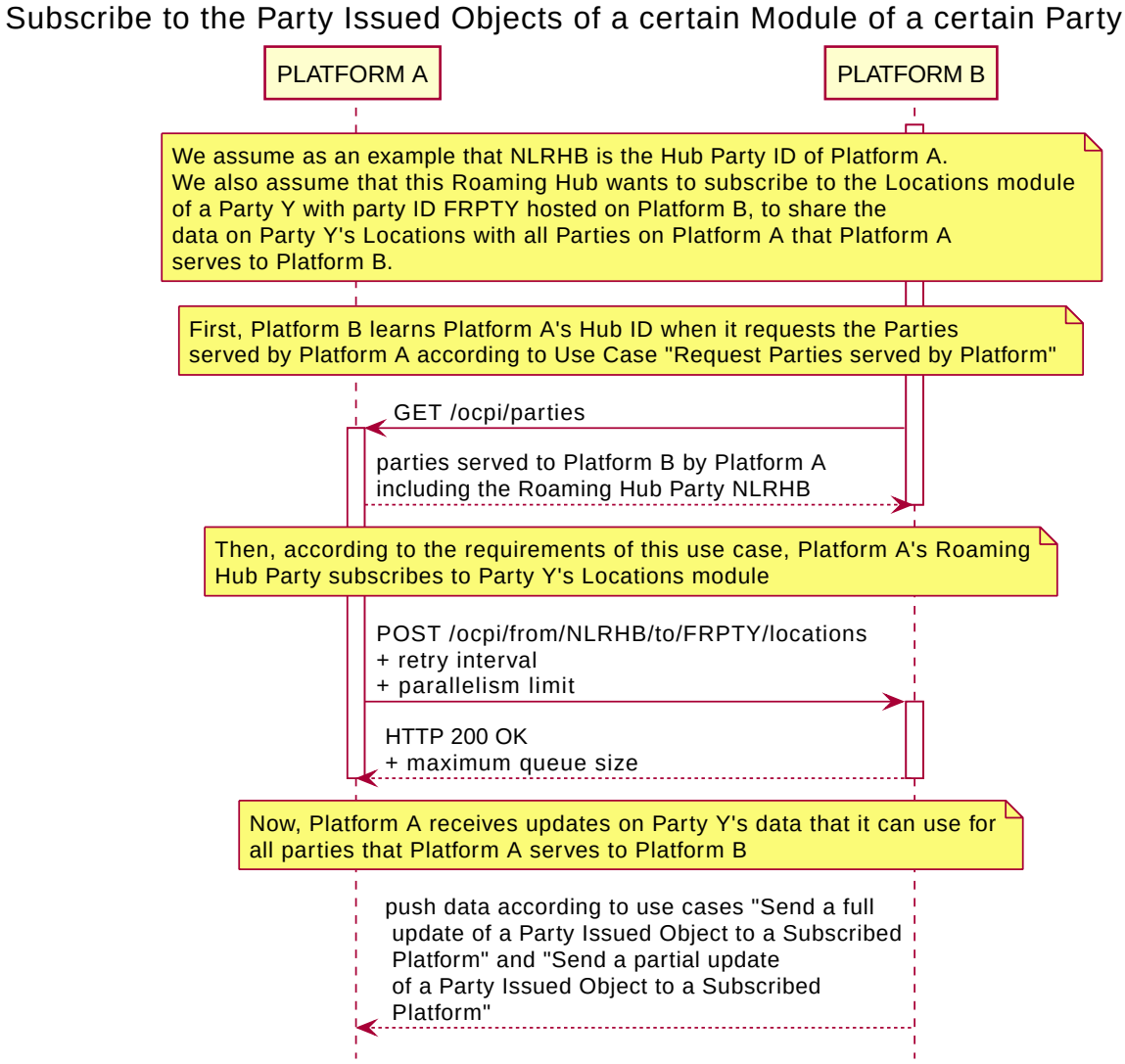


Figure 23. Sequence Diagram: Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub

Table 20. UC: 03.11 Requirements

ID	Precondition	Requirement
R.03.11.01		Platform A SHALL make the request to subscribe following Subscribe to Party Issued Objects of a certain Module of a certain Party.
R.03.11.02		Platform A SHALL use the Hub Party ID it provided to Platform B using the Request Parties served by Platform as the sender Party ID in the request URL according to Make a Request to a Party on behalf of a Party.
R.03.11.03	Platform B will not share Party Y's data of the Module given in the request with all Parties that Platform A serves to Platform B	Platform B SHALL send a response with the status_code field set to 5002
R.03.11.04	Platform B is willing to share Party Y's data of the Module given in the request with all Parties that Platform A serves to Platform B	Platform B MAY set up the subscription and notify Platform A of it according to Subscribe to Party Issued Objects of a certain Module of a certain Party

ID	Precondition	Requirement
R.03.11.05	Platform B set up a subscription according to R.03.11.04	Platform B MAY send updates of Party Issued Objects of Party Y to Platform A according to Send a full update of a Party Issued Object to a subscribed Platform, using Platform A's Hub Party ID as the receiver party ID in the request URLs.
R.03.11.06	Platform A receives updates of Party Issued Objects of Party Y addressed to its Hub Party ID as described in R.03.11.05	Platform A MAY use this data for all Parties that it serves to Platform B in exception to requirement the data privacy requirement for subscriptions.

3.2.12. UC: 03.12 - Subscribe to Party Issued Objects of a certain Module of a Hub

1	**Objective(s)**	**1.** Party X on Platform A starts receiving notifications about Party Issued Objects of all parties that Platform B serves to Platform A.
2	**Description**	When a Party with a self-hosted Platform connects to a Hub, they will typically want to receive the data of *all* other Parties that are connected to the Hub and that they have a contract with. This use case allows them to do that with a single subscription, instead of subscribing to each party on Platform B individually.
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, giving the Module that they wish to subscribe to and the Party ID of the Party that is subscribing, and giving the Hub Party as the Party ID to be subscribed to 2. Platform B responds, indicating if a subscription matching the request was created or not.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B offers Hub functionality to Platform A Platform A serves Party X to Platform B
6	**Postconditions**	Party X on Platform A is subscribed to the Roaming Hub of Platform B for the Module given in the request
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	In a typical scenario, Platform B would be a platform operated by a Roaming Hub and Platform A would be platform operated by a CPO, eMSP or similar self-hosting market party. This use case corresponds to the functionality called "GET All via Hubs" in OCPI 2.2.1 and 2.2 Typically, Hubs will give a self-hosting party connecting to them the ability to configure, in a web portal or such, which Parties the Hub's Platform serves to the connected self-hosted Platform. So indirectly the self-hosting company (Platform A) still decides whose data they will get pushed, although they have to trust the Roaming Hub (the operator of Platform B) to enforce their decisions.

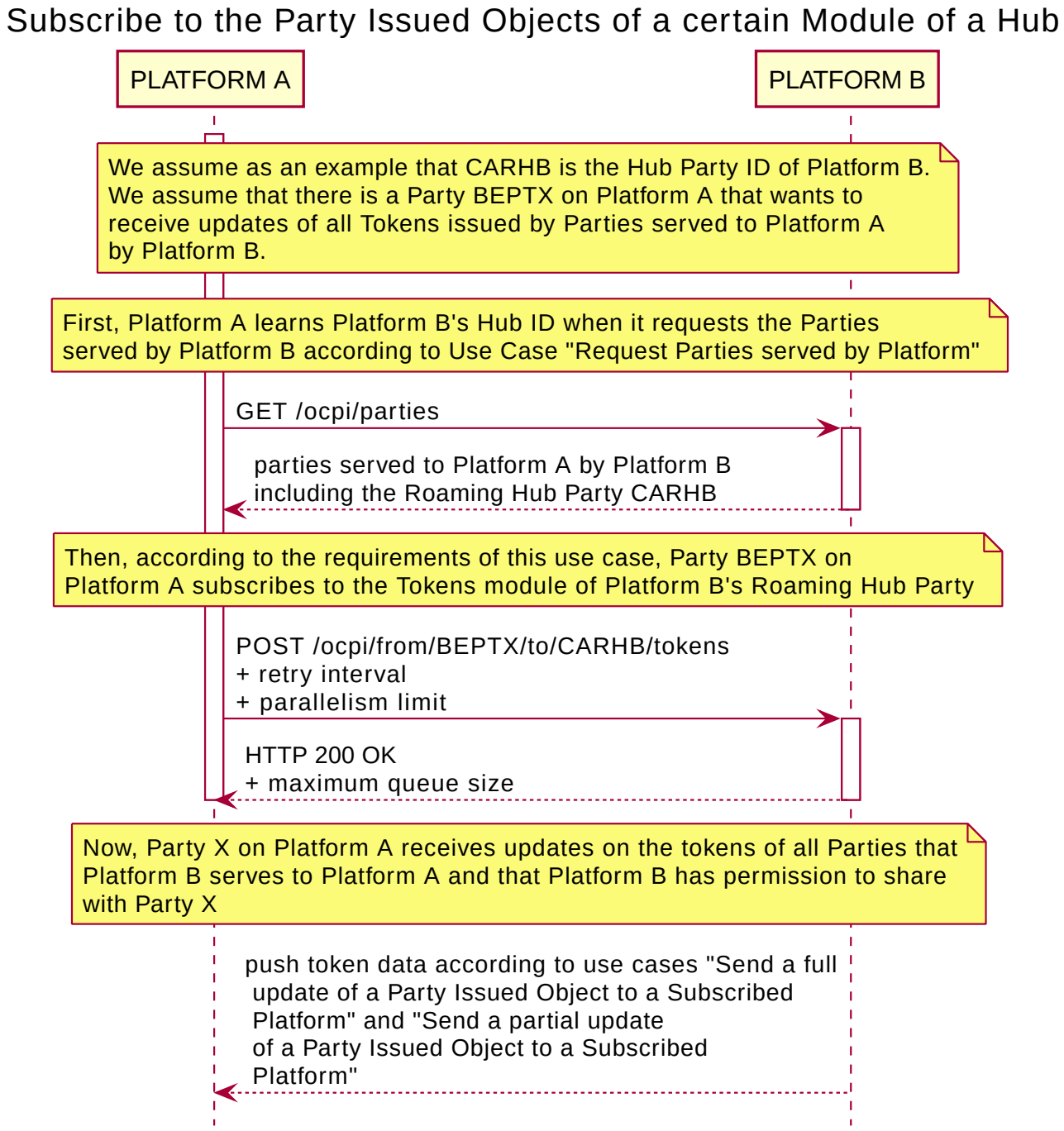


Figure 24. Sequence Diagram: Subscribe to Party Issued Objects of a certain Module of a Hub

Table 21. UC: 03.12 Requirements

ID	Precondition	Requirement
R.03.12.01		Platform A SHALL make the request to subscribe following Subscribe to Party Issued Objects of a certain Module of a certain Party.
R.03.12.02		Platform A SHALL use the Hub Party ID that Platform B provided to it using the Request Parties served by Platform as the receiver Party ID in the request URL according to Make a Request to a Party on behalf of a Party.

ID	Precondition	Requirement
R.03.12.03	Platform B set up a subscription in response to Platform A's request according to Subscribe to Party Issued Objects of a certain Module of a certain Party	Platform B MAY send, to Party X on Platfrom A, updates of Party Issued Objects of any Party that Platform B serves to Platform A according to Send a full update of a Party Issued Object to a subscribed Platform using Platform B's Hub Party ID as the sender party ID in the request URLs.

NOTE

R.03.12.03 entails that Platform B may send to Platform A updates of Party Issued Objects that were issued by a Party ID that Platform A has never encountered before. Platform A should be designed to handle such updates appropriately. Platform A may, for example, want to not use such objects in automated business processes for Party X until a human operator has confirmed that the object belongs to a Party that Party X has the appropriate contracts with.

3.2.13. UC: 03.13 - Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub

1	**Objective(s)**	**1.** Platform A can serve all Parties that Platform B serves to it to other Platforms
2	**Description**	Use cases 19 and 20 lend themselves to being used in combination and the possibility alone prompts the specification to say what should happen in that case. It indeed seems a useful possibility in case two Roaming Hubs, each operating their own Platform, would want to link up and open their roaming networks to each others' customers.
3	**Actors**	Roaming Hub
4	**Flow**	1. Platform A makes a request to Platform B, giving the Module that they wish to subscribe to, giving its own Roaming Hub ID as the Party ID of the Party that is subscribing, and giving Platform B's Hub Party ID as the Party ID of the Party to be subscribed to 2. Platform B responds, indicating if a subscription matching the request was created or not.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform A offers Hub functionality to Platform B Platform B offers Hub functionality to Platform A
6	**Postconditions**	The Roaming Hub of Platform A is subscribed to the Roaming Hub of Platform B for the Module given in the request
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	No equivalent functionality is available in the Roaming Hub support of OCPI 2.2.1 and 2.2 Roaming Hubs using this use case would have to guard against creating feedback loops of updates going back and forth. Proper implementation of update versioning should keep feedback loops from forming. Implementing routing tables might reduce even more unnecessary traffic in such a case.

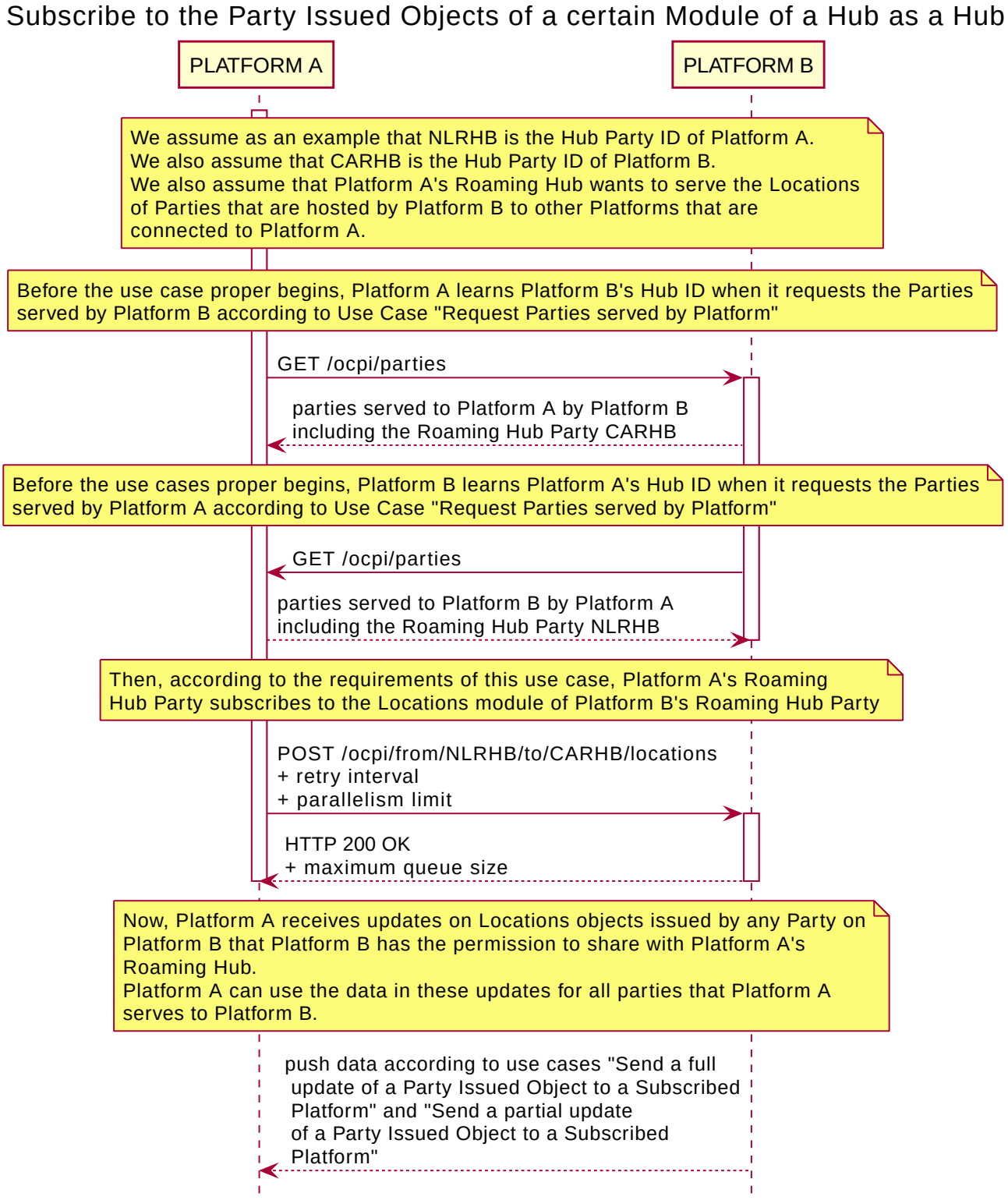


Figure 25. Sequence Diagram: Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub

Table 22. UC: 03.13 Requirements

ID	Precondition	Requirement
R.03.13.01		Platform A SHALL make the request to subscribe following Subscribe to Party Issued Objects of a certain Module of a certain Party.

ID	Precondition	Requirement
R.03.13.02		Platform A SHALL use the Hub Party ID that Platform B provided to it using the Request Parties served by Platform as the receiver Party ID in the request URL according to Make a Request to a Party on behalf of a Party.
R.03.13.03		Platform A SHALL use the Hub Party ID it provided to Platform B using the Request Parties served by Platform as the sender Party ID in the request URL according to Make a Request to a Party on behalf of a Party.
R.03.13.04	Platform B will not share the data of the Module given in the request of all Parties that it serves to Platform A with all Parties that Platform A serves to Platform B	Platform B SHALL send a response with the status_code field set to 5002
R.03.13.05	Platform B is willing to share the data of the Module given in the request of all Parties that it serves to Platform A with all Parties that Platform A serves to Platform B	Platform B MAY set up the subscription and notify Platform A of it according to Subscribe to Party Issued Objects of a certain Module of a certain Party
R.03.13.06	Platform B set up a subscription according to R.03.13.04	Platform B MAY send to Platform A updates of Party Issued Objects of any Party that it serves to Platform A according to Send a full update of a Party Issued Object to a subscribed Platform, using Platform A's Hub Party ID as the receiver party ID in the request URLs and using Platform B's Hub Party ID as the sender party ID in the request URLs.
R.03.13.07	Platform A receives updates of Party Issued Objects addressed to its Hub Party ID as described in R.03.13.06	Platform A MAY use this data for all Parties that it serves to Platform B in exception to requirement the data privacy requirement for subscriptions.

3.3. Object types for Party Issued Objects use cases

3.3.1. PartyIssuedObjectReference *class*

Property	Type	Cardinality	Description
id	CiAsciiString[1..39]	1	An identifier that uniquely identifies this Party Issued Object among other objects issued for the same module by the same party.

3.3.2. PartyIssuedObjectUpdate *class*

Property	Type	Cardinality	Description
issuer_party	PartyID	1	The party ID of the party that issued this object
id	CiAsciiString[1..39]	1	An identifier that uniquely identifies this Party Issued Object among other objects issued for the same module by the same party.
version	int	1	The version of the Party Issued Object that is in the `payload` field
payload	object	1	The representation of the Party Issued Object at the version given in the `version` field

3.3.3. SubscriptionCancellation *class*

Property	Type	Cardinality	Description
reason	SubscriptionCancellationReason	1	The reason for cancelling the subscription

3.3.4. SubscriptionCancellationReason *enum*

Value	Description
QUEUE_OVERFLOW	The number of queued updates has exceeded the maximum size of the persistent queue on the data sender Platform
DATA_NO_LONGER_AVAILABLE	The party that was subscribed to is no longer hosted on the Platform that was subscribed to, or they are no longer acting as a sender for the module that was subscribed to
OTHER	A generic code for any other reason for the data sender to cancel the subscription

3.3.5. SubscriptionParameterProposal *class*

Property	Type	Cardinality	Description
retry_interval	number	1	The proposed number of seconds that the sender Platform SHOULD wait before retrying a request for this subscription the first time

Property	Type	Cardinality	Description
parallelism_limit	int	?	The proposed maximum number of requests that are not yet responded to and have not yet timed out that the sender Platform is allowed to have made to the receiver Platform for this particular subscription at any one time. If this field is not given, the sender Platform SHALL act as if the value of 1 was provided.
max_queue_size	int	1	The proposed maximum number of updates to Party Issued Objects that the sender will queue for the receiver for this particular subscription

3.3.6. SubscriptionRenegotiationStatus *enum*

Value	Description
ACCEPTED	The proposed subscription parameters have been accepted and will take effect for subsequent Party Issued Object updates in the subscription.
REJECTED	The proposed subscription parameters have been rejected and will not take effect for subsequent Party Issued Object updates in the subscription.

3.3.7. SubscriptionRequest *class*

Property	Type	Cardinality	Description
retry_interval	number	1	The number of seconds that the sender Platform SHOULD wait before retrying a request for this subscription the first time
parallelism_limit	int	?	The maximum number of requests that are not yet responded to and have not yet timed out that the sender Platform is allowed to have made to the receiver Platform for this particular subscription at any one time. If this field is not given, the sender Platform SHALL act as if the value of 1 was provided.
max_queue_size	int	1	The maximum number of updates to Party Issued Objects that the sender promises to queue for the receiver for this particular subscription

3.3.8. SubscriptionResponse *class*

Property	Type	Cardinality	Description
max_queue_size	int	1	The maximum number of updates to Party Issued Objects that the sender promises to queue for the receiver for this particular subscription

3.3.9. SubscriptionState *class*

Property	Type	Cardinality	Description
retry_interval	number	1	The number of seconds that the sender Platform waits before retrying a request for this subscription the first time
current_queue_size	int	1	The number of updates to Party Issued Objects that the sender is storing in its persistent queue for this particular subscription at the time that this response is generated
max_queue_size	int	1	The maximum number of updates to Party Issued Objects that the sender promises to queue for the receiver for this particular subscription

4. Remote Procedure Calls

4.1. Introduction

This section is not normative.

Besides data replication, a lot of other functionality in OCPI is about remote procedure calls: one platform requesting an operation on another platform, expecting to receive a response from the other platform about the result of the operation.

Like for data replication, we give one specification of how all remote procedure calls in OCPI are done, to facilitate simple and correct implementations.

An RPC use case for OCPI 3.0 will specify:

- The purpose of the Remote Procedure Call, in the form of the use case header with its objectives, description, preconditions, postconditions etcetera
- An operation name
- An HTTP request verb to use
- If the Remote Procedure Call requires an HTTP request body, and if so, the form that the body should take
- The form for the OCPI response data
- A list of possible reasons for error with the OCPI response codes to use for those

4.2. Use Cases

4.2.1. UC: 04.01 - Make a Remote Procedure Call on behalf of a Party to another Party on another Platform

1	**Objective(s)**	**1.** Party X on Platform A gets Party Y on Platform B to perform an operation for them **2.** Party X on Platform A learns about the result of the requested operation
2	**Description**	Platform A sends an HTTP request to Platform B, stating which operation Party X wants performed and what all the parameters for the operation are. Platform B then responds with the result of the operation or an error message detailing why the operation could not be performed.
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, with the Module that they are requesting an operation of, the requested operation, and parameters for the operation. 2. Platform B responds with the result of the operation or an error message detailing why the operation could not be performed.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A has an operation name, an HTTP request verb and optionally a request payload for an operation that it wants to request on behalf of Party X from Party Y.

6	**Postconditions**	Party X has a response to its request, or has learned that Party Y on Platform B is not able to respond to it
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

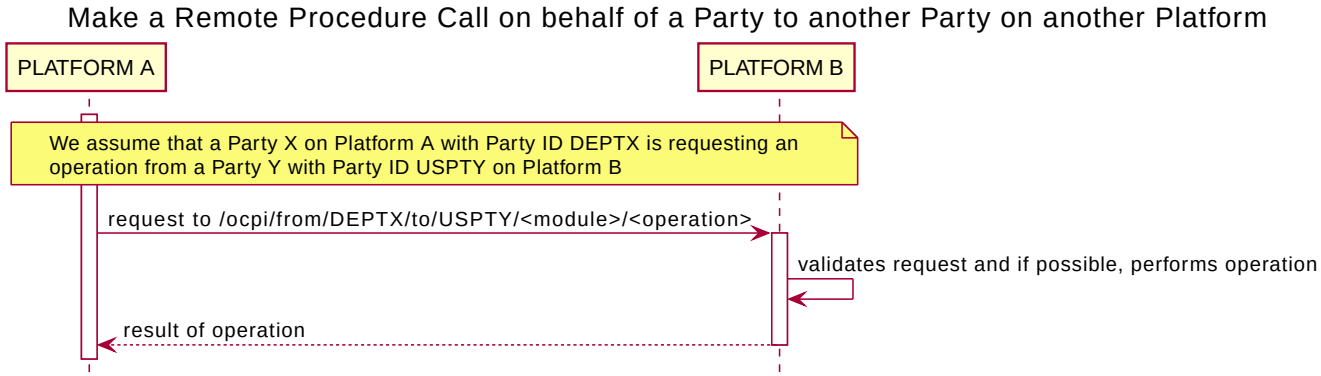


Figure 26. Sequence Diagram: Make a Remote Procedure Call on behalf of a Party to another Party on another Platform

Table 23. UC: 04.01 Requirements

ID	Precondition	Requirement
R.04.01.01		Platform A SHALL make its request to Platform B according to Make a request on behalf of a Party to a Party on another Platform
R.04.01.02		Platform A SHALL use a relative path for the request that consists of two segments, namely first the Module ID of the module that they are requesting an operation for, and second the operation name that is given in the use case that describes the operation later in this document.
R.04.01.03		The operation name used for R.04.01.2 SHALL NOT be one of "update", "cancel-my-subscription", "cancel-your-subscription" or "state" so as to prevent path conflicts with the Party Issued Objects use cases.
R.04.01.04		Platform A SHALL use the HTTP request verb that is given in the use case that describes the operation later in this document.
R.04.01.05	The use case for the operation, given later in this document, requires a request body	Platform A SHALL include a request body in the request, of the form required in that use case for the operation.
R.04.01.06	Platform B was able to execute the operation so that it produced a result	Platform B SHALL set the payload field of the OcpiResponse object in the response body to the result of the operation, in the form required in the use case for the operation

ID	Precondition	Requirement
R.04.01.07	Platform B failed to execute the operation for one of the reasons listed in the use case for the operation	Platform B SHALL set the status code of the OcpiResponse object in the response body to one of the error codes given for the reasons listed in the use case for the operation

4.2.2. UC: 04.02 - Let a Hub find a receiver Party for a Remote Procedure Call

1	**Objective(s)**	**1.** Let a Party get responses to Remote Procedure Calls even when they don't know which Party can respond to it
2	**Description**	Party X on Platform A makes a Remote Procedure Call request to the Roaming Hub Party on Platform B. Platform B sends a response, indicating the selected receiver Party in the response envelope.
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request to Platform B, setting the sender field to Party X's Party ID and the receiver field to Platform B's Roaming Hub Party ID 2. Platform B finds out which party can respond and obtains the response 3. Platform B sends the response to Platform A, indicating the selected receiver in the response envelope
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform A serves Party X to Platform B. Platform B has given a Roaming Hub Party ID to Platform A according to the *Request Parties served by Platform* use case. Platform A has an operation name, an HTTP request verb and optionally a request payload for an operation that it wants to request on behalf of Party X from any Party that Platform B serves to Platform A.
6	**Postconditions**	Party X has a response to its request, or has learned that no Party on Platform B is able to respond to it.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	This use case corresponds to the "Open Routing Request" concept of OCPI 2.2 and 2.2.1. The archetypical example of this use case is when a CPO seeks authorization to charge for a charge token that was swiped on one of its charging stations. When this CPO is connected to a Hub, it will want to send one request for authorization and check if the Hub can authorize it on behalf of any MSP hosted by that Hub.

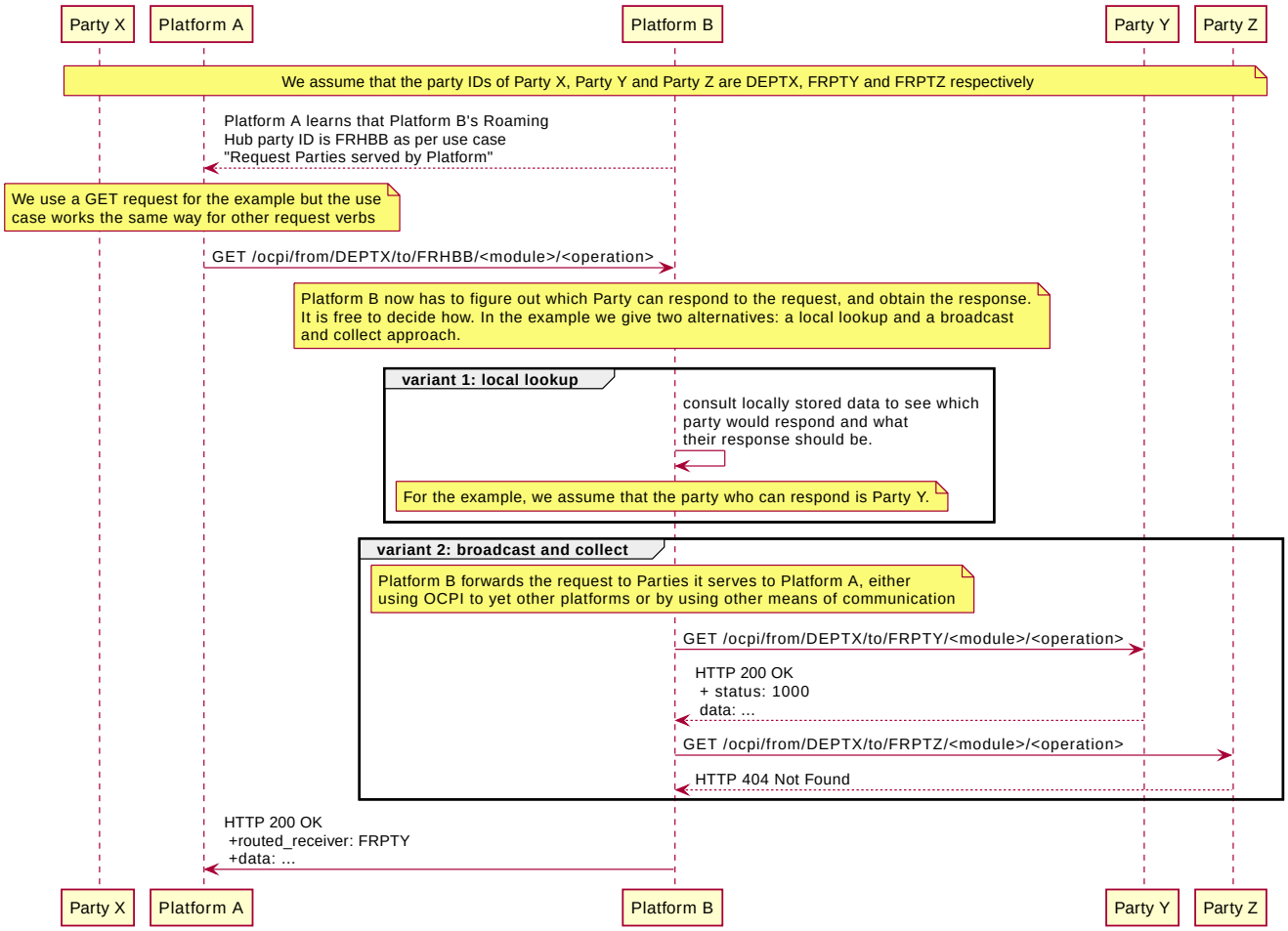


Figure 27. Sequence Diagram: Let a Hub find a receiver Party for a Remote Procedure Call

Table 24. UC: 04.02 Requirements

ID	Precondition	Requirement
R.04.02.01		Platform A SHALL request a Remote Procedure Call from Platform B according to Make a Remote Procedure Call on behalf of a Party ot another Party on another Platform
R.04.02.02		Platform A SHALL use Party X's Party ID as the sender party ID
R.04.02.03		Platform A SHALL use Platform B's Roaming Hub Party ID, as obtained using the *Request Parties served by Platform* use case, as the receiver party ID
R.04.02.05	Platform B is able to respond to this request with an OcpiResponse object and HTTP 200 or 201 status code on behalf of some Party it serves to Platform A	Platform B SHALL use that OcpiResponse object in the response body to Platform A, with the single modification that the `routed_receiver` field of the OcpiResponse object SHALL be set to the Party ID of the Party that Platform B is responding on behalf of
R.04.02.05	Platform B is not able to respond to this request with an OcpiResponse object or HTTP 200 or 201 status code on behalf of some Party it serves to Platform A	Platform B SHALL respond with HTTP status code 200 and an OcpiResponse object in the response body with the `status` field set to one of the Hub error codes

4.2.3. UC: 04.03 - Make a Remote Procedure Call Allowing Asynchronous Responses

1	**Objective(s)**	**1.** Let a Party get responses to Remote Procedure Calls even when they are not available immediately when the Party makes the request
2	**Description**	Party X on Platform A makes a Remote Procedure Call request to Party Y on Platform B. Platform A includes a callback identifier in the request to relate the eventual asynchronous response to it. Party Y on Platform A responds with an HTTP response indicating a preliminary status of request processing. When a final result of the request is available, Platform B posts this final result to an URL containing the callback ID from Platform A's original request.
3	**Actors**	Any
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B which receives it on behalf of Party Y. 2. Platform B answers to Platform A indicating a preliminary request processing status. 3. Platform B and Party Y execute the steps necessary to obtain a result to the operation requested by Platform A. 4. Platform B sends the final result to Platform A by making a second HTTP request to Platform A.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform A serves Party X to Platform B. Platform B serves Party Y to Platform A. Platform A has a module name, an operation name and optionally a request payload for an operation that it wants to request on behalf of Party X from Party Y.
6	**Postconditions**	Party X has a response to its request, or has learned that Party Y on Platform B is not able to respond to it.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	This use case is to be referenced by later use cases that require asynchronous remote procedure calls. These use cases have to specify the operation name, the request object type and the response object type to use.

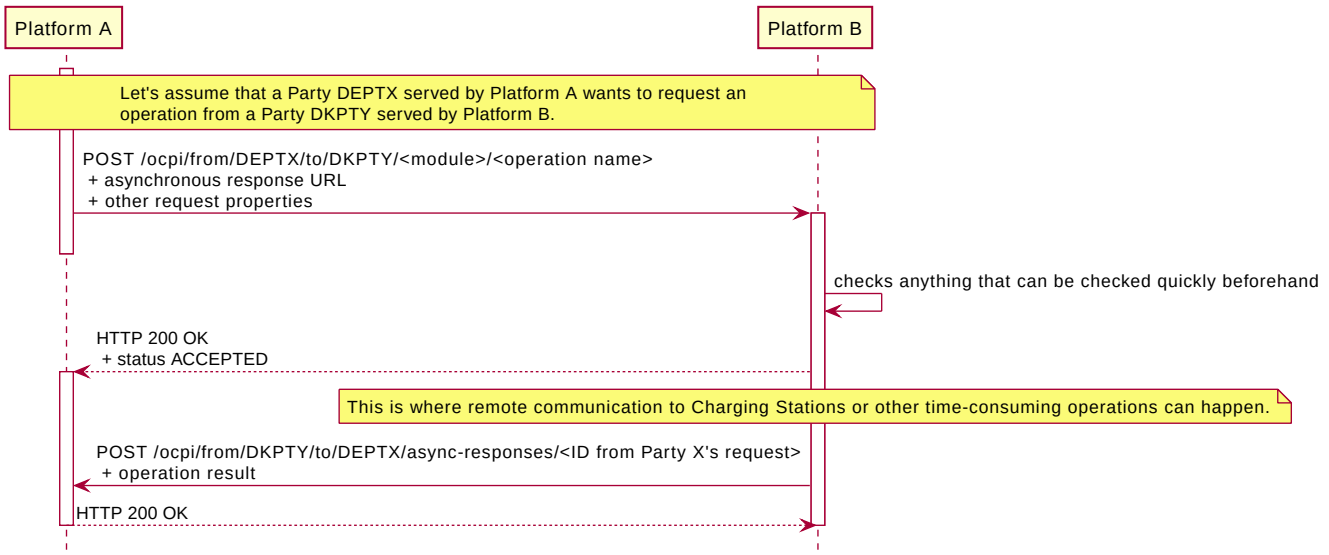


Figure 28. Sequence Diagram: Make a Remote Procedure Call Allowing Asynchronous Responses

Table 25. UC: 04.03 Requirements

ID	Precondition	Requirement
R.04.03.01		Platform A SHALL make its request to Platform B according to Make a Remote Procedure Call on Behalf of a Party to Another Party on Another Platform.
R.04.03.02		Platform A SHALL use the POST HTTP request verb.
R.04.03.03		Platform A SHALL use the module ID and operation given in a later use case referencing this one name when making its request to Platform B.
R.04.03.04		Platform A SHALL set the request body of its request to an AsyncRequest object.
R.04.03.05		Platform A SHALL set the value of the `callback_id` field in the AsyncRequest object to a value that was never used before for an asynchronous remote procedure call from Party X on Platform A to Party Y on Platform B.
R.04.03.06		Platform A SHALL fill the `payload` field in the AsyncRequest object as specified in a later use case referencing this use case.
R.04.03.07	Platform B receives Platform A's request and is able to process it	Platform B SHALL respond to Platform A's request as in R.04.01.06, setting the value of the `status_code` field of the OcpiResponse object to 1000, and putting an ImmediateResponseToAsyncRequest enum value in the `data` field of the OcpiResponse object.
R.04.03.08	Platform B accepted Platform A's remote procedure call, that is, Platform B responded to Platform A's request with an OcpiResponse object with 1000 as the value of the `status` field and ACCEPTED as the value of the `data` field	Platform B SHALL make a request to Platform A according to Make a request on behalf of a Party to a Party on another Platform to inform Platform A of the result of the remote procedure call once this result is available.

ID	Precondition	Requirement
R.04.03.09	Platform B is making the request to Platform A as required by R.04.03.08	Platform B SHALL use the POST HTTP request verb.
R.04.03.10	Platform B is making the request to Platform A as required by R.04.03.08	Platform B SHALL use a relative path for its request that consists of two segments, namely the string "async-responses" and the callback ID given in the AsyncRequest object in Platform A's request.
R.04.03.11	Platform B is making the request to Platform A as required by R.04.03.08	Platform B SHALL set the request body to an AsyncResponse that describes the result of the operation requested by Platform A.
R.04.03.12	Platform B is making the request to Platform A as required by R.04.03.08 and Platform B successfully completed the operation requested by Party X on Platform A	Platform B SHALL set the `result_type` of the AsyncResponse in the request body to SUCCESS and leave the `error` field of the AsyncResponse in the request body unset and set the `payload` field of the AsyncResponse object in the request body to a value specified in a later use case that references this use case.
R.04.03.13	Platform B is making the request to Platform A as required by R.04.03.08 and Platform B did not obtain a successful result of the operation requested by Party X on Platform A	Platform B SHALL set the `result_type` of the AsyncResponse in the request body to something else than SUCCESS and leave the `payload` field of the AsyncResponse object in the request body empty and set the `error` field of the AsyncRespnose to a value specified in a later use case that references this use case..
R.04.03.14	Platform A receives Platform B's request according to requirements R.04.03.08 to R.04.03.13	Platform A SHALL respond with an OcpiResponse object in the response body that leaves the `data` field unset and that has the `status_code` field set to 1000.

4.3. Object types for Remote Procedure Calls Use Cases

4.3.1. AsyncRequest *class*

Property	Type	Cardinality	Description
callback_id	AsciiString[1..36]	1	An identifier to relate a later asynchronous response to this request
payload	any JSON value	?	The parameters to the request to which the request sender expects an asynchronous response

4.3.2. AsyncResponse *class*

Property	Type	Cardinality	Description
result_type	AsyncResultType	1	The completion status of the requested asynchronous remote procedure call
error	any JSON value	?	The error that occurred during the execution of the asynchronous remote procedure call, if any
payload	any JSON value	?	The result of the asynchronous remote procedure call, if any

4.3.3. AsyncResultType *enum*

Value	Description
SUCCESS	Remote procedure call executed successfully
FAILED	Remote procedure call execution failed
NOT_SUPPORTED	The requested operation is not supported by the device that has to execute it
REJECTED	The requested operation was rejected by the device that has to execute it
TIMEOUT	No result was received from the device that had to execute the requested operation within a reasonable time

NOTE This corresponds to the CommandResultType enumeration of OCPI 2.2.1, with those values removed that were specific to certain commands, and with ACCEPTED renamed to SUCCESS.

4.3.4. ImmediateResponseToAsyncRequest *enum*

The intermediate status of an asynchronous remote procedure call that is given in the HTTP response to the initial request to make the call.

Value	Description
ACCEPTED	Remote procedure call accepted by the receiving Party.
NOT_SUPPORTED	This remote procedure call is not supported by the receiving Party or by the charging infrastructure device that has to execute it.
REJECTED	Remote procedure call was rejected by the receiving Party.

NOTE This corresponds to the the CommandResponseType enumeration of OCPI 2.2.1.

5. Locations

This chapter describes the Locations module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the Locations module, the type of data object is the Location. What a Location is is defined in the Terminology section of the Business Use Cases document. Essentially a Location is a Charging Pool plus extra data on how Drivers can use it and what amenities are available around the Pool. How a Location is represented in OCPI messages is specified below.

Initially a Locations module was conceived in earlier OCPI versions to inform an MSP of where their Drivers can charge. Nowadays, it can be used also by other actors than MSPs who want to know about a CPO's physical charging infrastructure for other purposes. One can think of SCSPs who want to know which physical infrastructure there is for them to set charging profiles on, or navigation service providers who want to display charging facilities to their users in an interactive maps app.

5.1. Changes from OCPI 2.2.1

- Removed the country code and party ID, as from all Party Issued Objects, because these are now unambiguously transferred using the Party Issued Object replication mechanism
- Split off the replication of live EVSE status to another module
- Moved the commands found in 2.2.1 that operate on Locations from the Commands module to the Locations module
- Added a Charging Station layer between the Location and EVSE layers of the Locations data model
- Added a Parking object to each EVSE to describe the parking situation
- Added a `max_power` field to the Location object so that CPOs can disclose how much power a Location as a whole can draw.
- Grouped address fields inside a Location and made them optional.
- Added MCS and SAE J3400 Connector types.
- Added TAXI_ONLY parking restriction.
- Added capabilities field on Connector to reflect IEC 15118 compatibility.
- Added a field for an assistance telephone number to the Location object.
- The REMOTE_START_STOP_CAPABLE capability was split in separate capabilities for start and stop, so that CPOs can allow remote start without allowing remote stop.

5.2. Replicating Location objects

5.2.1. UC: 05.01 - Replicate Location objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains and maintains an up-to-date copy of the Location objects that are issued by Party Y on a Platform B
2	**Description**	Using the use cases of the Party-Issued Objects chapter, Location objects are replicated from Party Y to Party X

3	**Actors**	eMSP, CPO, NAP, NSP
4	**Flow**	1. Party X subscribes to Party Y's Location objects 2. Party Y pushes all their Location objects as of subscription time to Party X 3. Party Y pushes every newly updated Location object to Party X as soon as these updates happen 4. This continues until either party cancels the subscription
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Locations to Party X on Platform A.
6	**Postconditions**	Party X has up-to-date information on the charging infrastructure offered by Party Y
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 26. UC: 05.01 Requirements

ID	Precondition	Requirement
R.05.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "locations" as the ModuleID value.
R.05.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.05.01.01
R.05.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.05.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the Location object type.
R.05.01.04		Private Charging Stations, home or business charging facilities that do not need to be published on public apps, and do not require remote control via OCPI, SHOULD NOT be issued through the OCPI Locations module.

ID	Precondition	Requirement
R.05.01.05	Party X receives a Location with the `publish` field set to `false`	Party X SHALL NOT allow this Location to be shown in an app or on a website or such unless it is to the holder of a Token in the `publish_allowed_to` list. Even parties like NSP or eMSP that did not issue this Token MAY show this Location on an app or website, but only to the holder of that Token. Such NSPs or eMSPs MAY show the Location if and only if the user of their app has provided information about their Token and that information matches all the fields of one of the PublishToken tokens in the list.
R.05.01.06		Party Y SHOULD make sure that the `max_electric_power`, `max_voltage`, `max_amperage` and `power_type` fields in the Connector objects in the Location objects that they send to Party X are filled in so that all static constraints on the charging power are taken into account. This means that Party Y should not only take into account constraints that are present in the charging equipment as it leaves the factory, but also things like constraints on the power supply to the charging equipment and constraints introduced by damage to the equipment. It also means that short-term changes in expected power output, typically lasting less than 24 hours, due to things like energy trading and load balancing between multiple EVSEs at the same location, do not have to be taken into account when filling in these fields.

5.2.2. Example Location objects

5.2.2.1. Example public Location

This is an example of a public Location. Can be used by any EV Driver as long as their eMSP has a roaming agreement with the CPO or the Location has an ad-hoc payment possibility

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


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

```
    }
  },
  "parking_type": "ON_STREET",
  "charging_pool": [
    {
      "id": "CH0023",
      "capabilities": [
        "RESERVABLE"
      ],
      "floor_level": "-1",
      "evses": [
        {
          "uid": "3256",
          "evse_id": "BE*BEC*E041503001",
          "connectors": [
            {
              "id": "1",
              "standard": "IEC_62196_T2",
              "format": "CABLE",
              "power_type": "AC_3_PHASE",
              "max_voltage": 220,
              "max_amperage": 16,
              "last_updated": "2015-03-16T10:10:02Z"
            },
            {
              "id": "2",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 220,
              "max_amperage": 16,
              "last_updated": "2015-03-18T08:12:01Z"
            }
          ],
          "physical_reference": "1",
          "parking": {
            "vehicle_types": ["MOTORCYCLE", "PERSONAL_VEHICLE"],
            "drive_through": false,
            "evse_position": "RIGHT",
            "restricted_to_type": false,
            "reservation_required": false
          },
          "last_updated": "2015-06-28T08:12:01Z"
        },
        {
          "uid": "3257",
          "evse_id": "BE*BEC*E041503002",
          "connectors": [
            {
              "id": "1",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 220,
              "max_amperage": 16,
              "last_updated": "2015-06-29T20:39:09Z"
            }
          ],
          "physical_reference": "2",
          "parking": {
            "vehicle_types": ["MOTORCYCLE", "PERSONAL_VEHICLE"],
            "drive_through": false,
            "evse_position": "RIGHT",
            "restricted_to_type": false,
            "reservation_required": false
          },
          "last_updated": "2015-06-29T20:39:09Z"
        }
      ],
      "last_updated": "2015-06-29T20:39:09Z"
```

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

5.2.2.2. Example destination Location

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

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

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

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

```
{
  "id": "3e7b39c2-10d0-4138-a8b3-8509a25f9920",
  "publish": true,
  "name": "Infuse",
  "address": {
    "address": "Tamboerijn 7",
    "city": "Etten-Leur",
    "postal_code": "4876 BS",
    "country": "NLD",
    "coordinates": {
      "latitude": "51.562659",
      "longitude": "4.638865"
    }
  },
  "parking_type": "PARKING_LOT",
  "charging_pool": [
    {
      "id": "IHOMER001",
      "coordinates": {
        "latitude": "51.562794",
        "longitude": "4.638964"
      },
      "evses": [
        {
          "uid": "fd855359-bc81-47bb-bb89-849ae3dac89e",
          "evse_id": "NL*ALF*E000000001",
          "connectors": [
            {
              "id": "1",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 220,
              "max_amperage": 16,
              "last_updated": "2019-07-01T12:12:11Z"
            }
          ],
          "parking": {
```

```
            "vehicle_types": ["MOTORCYCLE", "PERSONAL_VEHICLE"],
            "drive_through": false,
            "evse_position": "HEAD",
            "restricted_to_type": false,
            "reservation_required": false,
            "parking_restrictions": [
              {
                "group": "CUSTOMERS",
                "applies_outside_opening_hours": true
              }
            ]
          },
          "last_updated": "2019-07-01T12:12:11Z"
        },
        {
          "uid": "9c6aab5e-7a59-4300-9c7e-996642e6fd82",
          "evse_id": "NL*ALF*E000000002",
          "connectors": [
            {
              "id": "2",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 220,
              "max_amperage": 16,
              "last_updated": "2019-06-30T12:13:14Z"
            }
          ],
          "parking": {
            "vehicle_types": ["MOTORCYCLE", "PERSONAL_VEHICLE"],
            "drive_through": false,
            "evse_position": "HEAD",
            "restricted_to_type": false,
            "reservation_required": false,
            "parking_restrictions": [
              {
                "group": "CUSTOMERS",
                "applies_outside_opening_hours": true
              }
            ]
          },
          "last_updated": "2019-06-30T12:13:14Z"
        }
      ],
      "last_updated": "2019-07-01T12:12:11Z"
    },
    {
      "id": "IHOMER002",
      "coordinates": {
        "latitude": "51.562769",
        "longitude": "4.638906"
      },
      "evses": [
        {
          "uid": "0d7a4e3a-1100-4014-bfde-62bac6ad8471",
          "evse_id": "NL*ALF*E000000003",
          "connectors": [
            {
              "id": "1",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 220,
              "max_amperage": 16,
              "last_updated": "2019-06-30T11:00:00Z"
            }
          ],
          "parking": {
            "vehicle_types": ["MOTORCYCLE", "PERSONAL_VEHICLE"],
            "drive_through": false,
```

```
            "evse_position": "HEAD",
            "restricted_to_type": false,
            "reservation_required": false,
            "parking_restrictions": [
              {
                "group": "CUSTOMERS",
                "applies_outside_opening_hours": true
              }
            ]
          },
          "last_updated": "2019-06-30T11:00:00Z"
        },
        {
          "uid": "21110f8c-3f71-43dc-b42b-e16d7260a7c1",
          "evse_id": "NL*ALF*E000000004",
          "connectors": [
            {
              "id": "2",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 220,
              "max_amperage": 16,
              "last_updated": "2019-06-28T16:53:12Z"
            }
          ],
          "parking": {
            "vehicle_types": ["MOTORCYCLE", "PERSONAL_VEHICLE"],
            "drive_through": false,
            "evse_position": "HEAD",
            "restricted_to_type": false,
            "reservation_required": false,
            "parking_restrictions": [
              {
                "group": "CUSTOMERS",
                "applies_outside_opening_hours": true
              }
            ]
          },
          "last_updated": "2019-06-28T16:53:12Z"
        }
      ],
      "last_updated": "2019-06-30T11:00:00Z"
    }
  ],
  "time_zone": "Europe/Amsterdam",
  "last_updated": "2019-07-01T12:12:11Z"
}
```


5.2.2.3. Example destination Locations not published, but paid guest usage possible

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

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

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

- `publish` = `false`
- `publish_allowed_to` is not used

- `parking_type` is not used`
- `EVSE.parking.parking_restrictions` = `CUSTOMERS` May still be useful so a support desk can also tell this to a customer.


```
{
  "id": "3e7b39c2-10d0-4138-a8b3-8509a25f9920",
  "publish": false,
  "name": "Infuse",
  "address": {
    "address": "Tamboerijn 7",
    "city": "Etten-Leur",
    "postal_code": "4876 BS",
    "country": "NLD",
    "coordinates": {
      "latitude": "51.562659",
      "longitude": "4.638865"
    }
  },
  "charging_pool": [
    ... elided for brevity ...
  ],
  "time_zone": "Europe/Amsterdam",
  "last_updated": "2019-07-01T12:12:11Z"
}
```


5.2.2.4. Example Location with limited visibility

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

Typical examples where this is useful:

- Charging Stations in the parking garage of an apartment building. Only owners can see/control the Charging Stations.
- Charging Stations at an office, for employees only. Only employees can see/control the Charging Stations.
- Charging Stations at vehicle depot. Any employee can see/control an Charging Station, even transaction they did not start. Use `group_id` for this.

The Locations will not be published to the general public. Only selected Tokens can see (and control) the Charging Stations via eMSP app.

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


```
{
  "id": "f76c2e0c-a6ef-4f67-bf23-6a187e5ca0e0",
  "publish": false,
  "publish_allowed_to": [{
    "visual_number": "12345-67",
    "issuer": "NewMotion"
  }, {
    "visual_number": "0055375624",
    "issuer": "ANWB"
  }, {
    "uid": "12345678905880",
    "type": "RFID"
  }],
```

```
  "name": "Water State",
  "address": {
    "address": "Taco van der Veenplein 12",
    "city": "Leeuwarden",
    "postal_code": "8923 EM",
    "country": "NLD",
    "coordinates": {
      "latitude": "53.213763",
      "longitude": "5.804638"
    }
  },
  "parking_type": "UNDERGROUND_GARAGE",
  "charging_pool": [
    {
      "id": "DEV000013",
      "evses": [
        {
          "uid": "8c1b3487-61ac-40a7-a367-21eee99dbd90",
          "evse_id": "NL*ALL*EGO0000013",
          "connectors": [
            {
              "id": "1",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 230,
              "max_amperage": 16,
              "last_updated": "2019-09-27T00:19:45Z"
            }
          ],
          "parking": {
            "vehicle_types": ["MOTORCYCLE", "PERSONAL_VEHICLE"],
            "drive_through": false,
            "evse_position": "HEAD",
            "restricted_to_type": false,
            "reservation_required": false
          },
          "last_updated": "2019-09-27T00:19:45Z"
        }
      ],
      "last_updated": "2019-09-27T00:19:45Z"
    }
  ],
  "time_zone": "Europe/Amsterdam",
  "last_updated": "2019-09-27T00:19:45Z"
}
```


5.2.2.5. Example private Charging Station with eMSP app control

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

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

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


```
{
  "id": "a5295927-09b9-4a71-b4b9-a5fffdfa0b77",
  "publish": false,
  "publish_allowed_to": [{
```

```
    "visual_number": "0123456-99",
    "issuer": "MoveMove"
  }],
  "parking_type": "ON_DRIVEWAY",
  "charging_pool": [
    {
      "id": "DEV000001",
      "evses": [
        {
          "uid": "4534ad5f-45be-428b-bfd0-fa489dda932d",
          "evse_id": "DE*ALL*EGO0000001",
          "connectors": [
            {
              "id": "1",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_1_PHASE",
              "max_voltage": 230,
              "max_amperage": 8,
              "last_updated": "2019-04-05T17:17:56Z"
            }
          ],
          "parking": {
            "vehicle_types": ["PERSONAL_VEHICLE"],
            "drive_through": false,
            "evse_position": "RIGHT",
            "restricted_to_type": false,
            "reservation_required": false
          },
          "last_updated": "2019-04-05T17:17:56Z"
        }
      ],
      "last_updated": "2019-04-05T17:17:56Z"
    }
  ],
  "time_zone": "Europe/Berlin",
  "last_updated": "2019-04-05T17:17:56Z"
}
```


5.2.2.6. Example Charging Station in a parking garage with opening hours

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

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

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


```
{
  "id": "cbb0df21-d17d-40ba-a4aa-dc588c8f98cb",
  "publish": true,
  "name": "P-Huset Leonard",
  "address": {
    "address": "Claesgatan 6",
    "city": "Malmö",
    "postal_code": "214 26",
    "country": "SWE",
    "coordinates": {
      "latitude": "55.590325",
```

```
      "longitude": "13.008307"
    }
  },
  "parking_type": "PARKING_GARAGE",
  "charging_pool": [
    {
      "id": "DEV00000012",
      "evses": [
        {
          "uid": "eccb8dd9-4189-433e-b100-cc0945dd17dc",
          "evse_id": "SE*EVC*E000000123",
          "connectors": [
            {
              "id": "1",
              "standard": "IEC_62196_T2",
              "format": "SOCKET",
              "power_type": "AC_3_PHASE",
              "max_voltage": 230,
              "max_amperage": 32,
              "last_updated": "2017-03-07T02:21:22Z"
            }
          ],
          "parking": {
            "vehicle_types": [
              "MOTORCYCLE",
              "PERSONAL_VEHICLE"
            ],
            "drive_through": false,
            "evse_position": "HEAD",
            "restricted_to_type": false,
            "reservation_required": false
          },
          "last_updated": "2017-03-07T02:21:22Z"
        }
      ],
      "last_updated": "2017-03-07T02:21:22Z"
    }
  ],
  "time_zone": "Europe/Stockholm",
  "opening_times": {
    "twentyfourseven": false,
    "regular_hours": [{
      "weekday": 1,
      "period_begin": "07:00",
      "period_end": "18:00"
    }, {
      "weekday": 2,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 3,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 4,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 5,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 6,
      "period_begin": "07:00",
      "period_end": "18:00"
    },{
      "weekday": 7,
      "period_begin": "07:00",
      "period_end": "18:00"
    }]
```

```
  },
  "charging_when_closed": true,
  "last_updated": "2017-03-07T02:21:22Z"
}
```

A formal version of the data schema will be made available as JSON Schema ([JSONSCHEMA])separately.

5.3. Remote Procedure Calls on Location objects

5.3.1. UC: 05.02 - Reserve an EVSE at a Location

1	**Objective(s)**	**1.** Party Y reserves an EVSE issued by them for a Charging Session paid for by Party X **2.** Party X knows which EVSE has been reserved for them and for which time window
2	**Description**	An asynchronous remote procedure call is made by Party X to Party Y. In the request Party X sends the specifics of the reservation that they wish to make. In the response Party Y sends a confirmation or a rejection of the reservation.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the charge token that reservation is for, the time until which the reserved EVSE is to be held, and the ID of the Location on which an EVSE should be reserved. 2. Party Y reserves the EVSE. 3. Platform B makes a request on behalf of Party Y to Platform A receiving the request on behalf of Party X. This request informs Party X of whether the reservation succeeded.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Locations to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to reserve an EVSE at one of Party Y's Locations. * Party Y reserved an EVSE at one of Party Y's Locations for a charging session with the Charge Token that Party X sent them in the request. Also Party X knows that their reservation request was turned into a reservation and that they will have to pay for a Charging Session started with the given Token during the reservation.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party. If Party Y received the request and reserved an EVSE for Party X, but Platform B failed to deliver a response to Platform A for Party X, then there is a reservation but Party X does not know about it. TODO how to prevent unwanted no-show charges on Party X in this case?
8	**Remark(s)**	

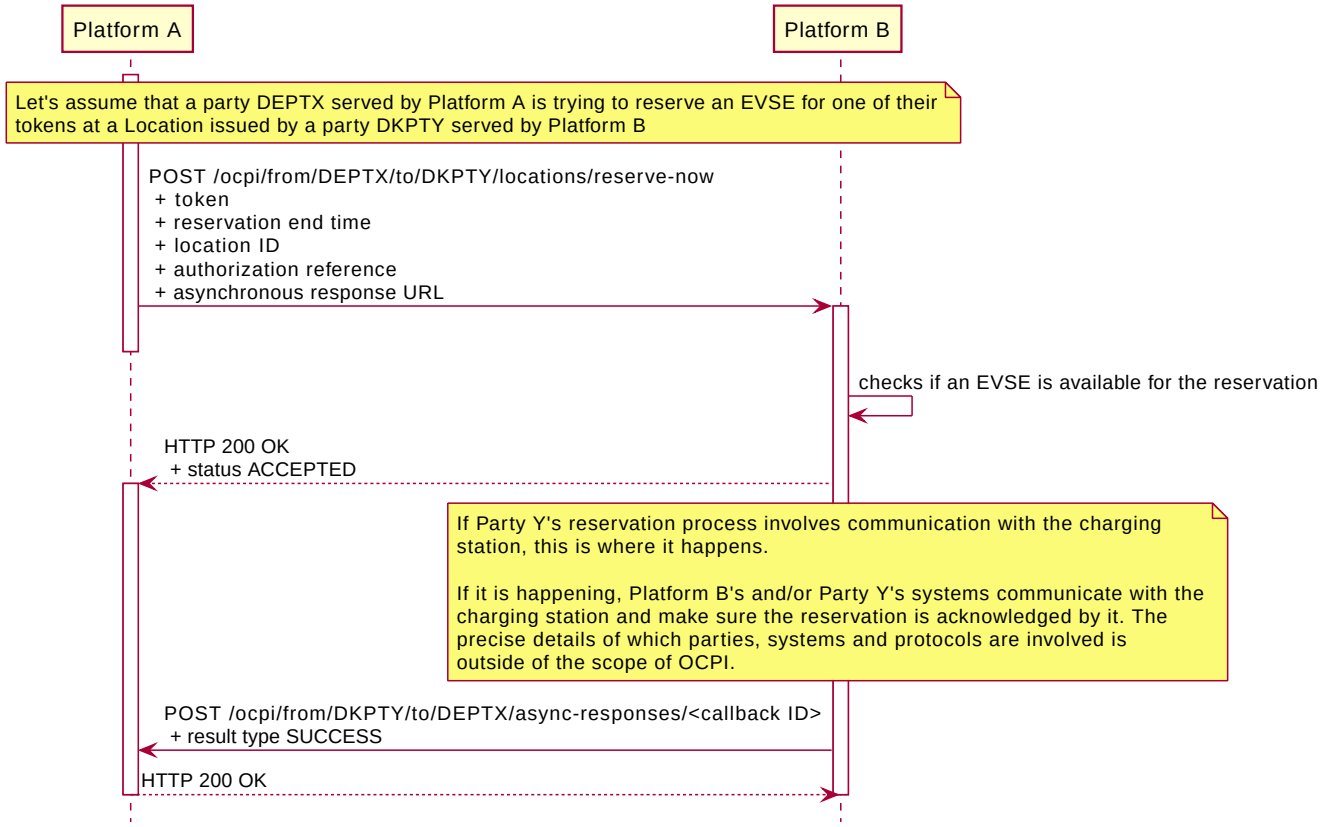


Figure 29. Sequence Diagram: Reserve an EVSE at a Location

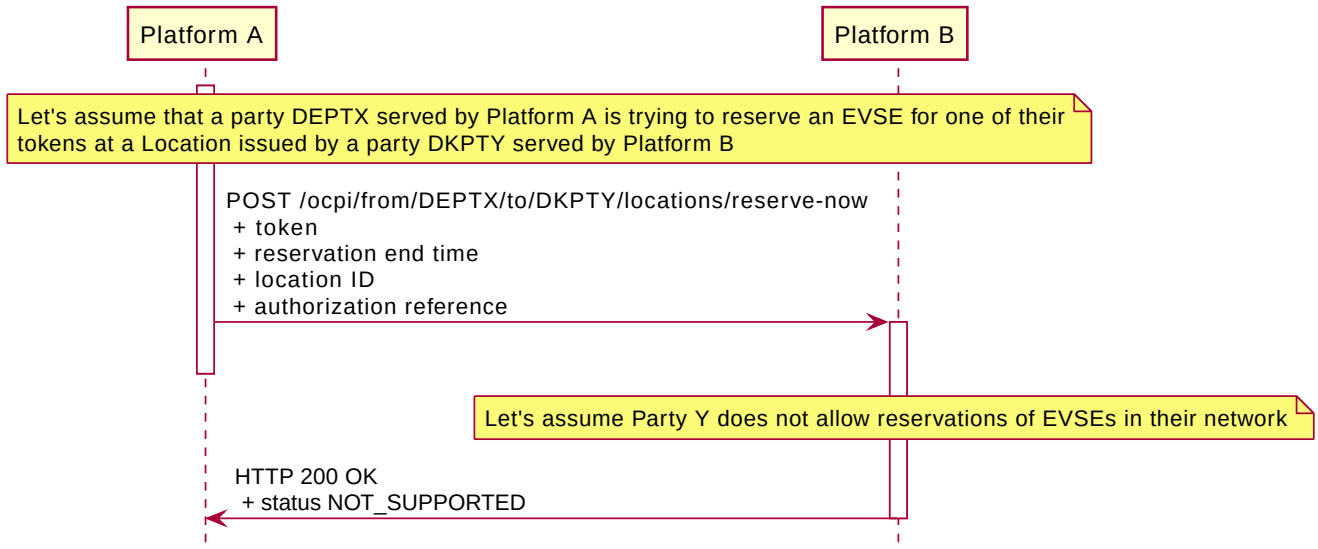


Figure 30. Sequence Diagram: Reserve an EVSE at a Location - request rejected by Party Y's backend

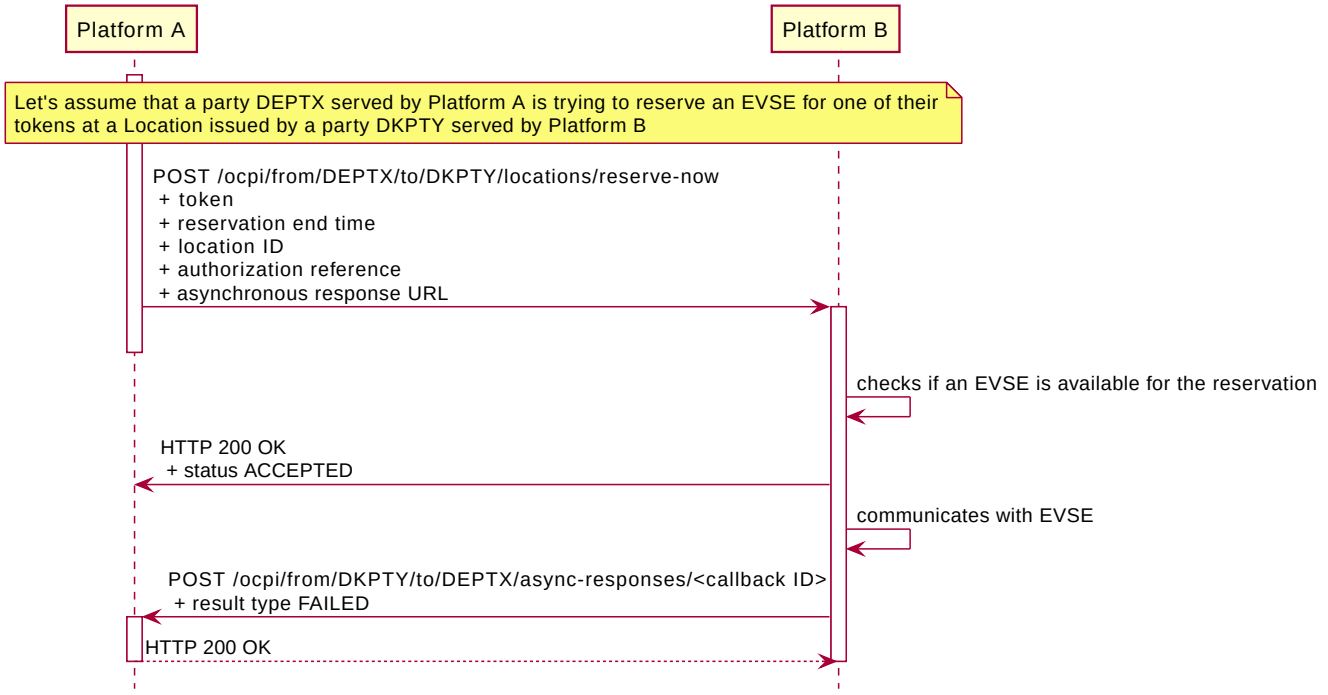


Figure 31. Sequence Diagram: Reserve an EVSE at a Location - request rejected by Charging Station

Table 27. UC: 05.02 Requirements

ID	Precondition	Requirement
R.05.02.01		Platform A SHALL make the request to reserve an EVSE following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.02.02		Platform A SHALL use "reserve-now" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.02.03		Platform A SHALL use a ReserveNowRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.02.04	Platform A does not mean to update an existing reservation	Platform A SHALL use a reservation identifier that was never used before for a reservation by Party X on a Location issued by Party Y.
R.05.02.05	No EVSE UID is given in the request by Platform A	Party Y SHOULD treat the request as a request to keep one EVSE available in the Location with the ID given by Party X in the request for a Charging Session with the given Token until the end time given in Platform A's reservation request.
R.05.02.06	An EVSE UID is given in the request by Platform A	Party Y SHOULD treat the request as a request to keep the EVSE with the given UID available for a Charging Session with the given Token until the end time given in Platform A's reservation request.
R.05.02.07	Party Y already reserved an EVSE for Party X with the reservation ID and Location ID given in the request from Platform A	Party Y SHOULD remove the old reservation with that reservation ID and Location ID, and then process the request for a reservation.

ID	Precondition	Requirement
R.05.02.08	Party Y cannot fulfill the request from Party X because of the state of the reservation or the status of the EVSE	Platform B SHALL respond to Platform A's request with ACCEPTED in the `data` field of the OcpiResponse in the response body. Platform B SHALL then send an asynchronous response with the result_type field of the AsyncResponse set to REJECTED and the error field of the AsyncResponse set to an appropriate ReservationError value.
R.05.02.09	Party Y reserved an EVSE as requested by Party X	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse set to SUCCESS and the `error` and `payload` fields of the AsyncResponse both left unset.
R.05.02.10	Platform A delivered an asynchronous response with result type SUCCESS	Party Y SHALL have an EVSE available at the Location with the identifier given in Platform A's request for charging with the Token given in Platform A's request until the time given in Platform A's request
R.05.02.11	Party Y has responded asynchronously with result type SUCCESS and the token type of the Token in Platform A's request is AD_HOC_USER or APP_USER	The reservation counts as authorization for charging as specified in the reservation request. That is, Party Y SHALL authorize attempts to start a Charge Session with the token given in Platform A's request on the EVSE reserved for this request without going through real-time authorization until the end time given in Platform A's request.
R.05.02.12	Party Y has responded asynchronously with result type SUCCESS and the token type of the Token in Platform A's request is RFID	The reservation counts as authorization for charging as specified in the reservation request for a duration of 15 minutes. That is, Party Y SHALL authorize attempts to start a Charge Session with the token given in Platform A's request on the EVSE reserved for this request without real-time authorization until either 15 minutes after it received the reservation request or the end time given in Platform A's request, whichever comes earlier.

NOTE An unused Reservation of an EVSE may result in cost being incurred by Party X, thus also in a Session and a CDR being replicated from Party Y to Party X.

NOTE The reservation ID used in Platform A's request is only required to be unique among reservations made by Party X on Party Y's infrastructure. If Platform A passes the reservation on to other systems, like to the Charging Station with OCPP, it has to make sure that it uses a reservation ID that is unique in those other systems. Therefore, Platform A will typically generate its own reservation IDs that are unique on the entire Platform.

NOTE There is no requirement that Tokens used by Party X for this use case be previously replicated to Party Y with the Tokens module.

5.3.2. UC: 05.03 - Cancel a Reservation as an eMSP

1	**Objective(s)**	**1.** Party Y will no longer keep an EVSE available for charging with a certain Token from Party X
2	**Description**	An eMSP may learn from a Driver that a certain reservation is not needed anymore. The eMSP may also conclude by itself that a certain reservation is no longer desirable, e.g. when a Driver for whom a reservation was made runs out of prepaid credit. To cancel a reservation, the eMSP (Party X) makes an asynchronous remote procedure call to the CPO (Party Y). In the request Party X sends the identifier of the reservation that they wish to cancel. In the response Party Y sends a confirmation or a rejection of the cancelation request.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the identifier of the reservation that is to be canceled. 2. Party Y determines if it is willing and able to execute the requested cancelation. 3. Platform B responds to Platform A on behalf of Party Y to inform Party X if their request was accepted. 4. If the request was accepted, Party Y proceeds to remove the reservation from all affected systems, potentially including a Charging Station. 5. If the request was accepted, Platform B sends a request to Platform A on behalf of Party Y to inform Party X of the result of the cancelation operation.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Locations to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to cancel a reservation of an EVSE at one of Party Y's Locations. * Party Y attempted to cancel the reservation at Party X's request. Also Party X knows whether this attempt was successful or not.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party. If Party Y received the request and canceled the reservation for Party X, but Platform B failed to deliver an asynchronous response to Platform A for Party X, then the reservation was canceled but Party X does not know about it. There is no automatic remediation process for this condition.
8	**Remark(s)**	

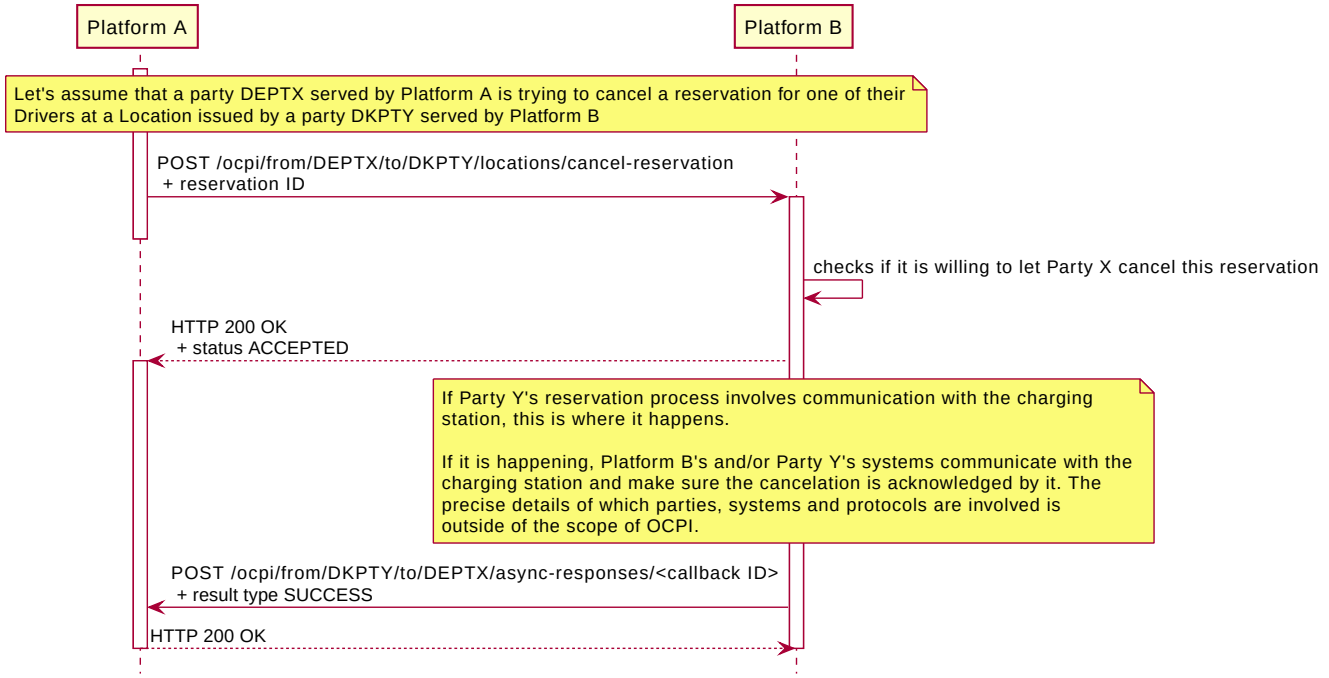


Figure 32. Sequence Diagram: Cancel a Reservation as an eMSP

Table 28. UC: 05.03 Requirements

ID	Precondition	Requirement
R.05.03.01		Platform A SHALL make the request to cancel a reservation following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.03.02		Platform A SHALL use "cancel-reservation" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.03.03		Platform A SHALL use a CancelReservationRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.03.04		Platform A SHALL set the `reservation_id` field of theCancelReservationRequest to a reservation identifier that it previously used before to make or update a reservation according to Reserve an EVSE at a Location.
R.05.03.05	Party Y cannot fulfill the request from Party X because of the state of the reservation or the status of the EVSE	Platform B SHALL respond to Platform A's request with ACCEPTED in the `data` field of the OcpiResponse in the response body. Platform B SHALL then send an asynchronous response with the result_type field of the AsyncResponse set to REJECTED and the error field of the AsyncResponse set to an appropriate ReservationError value.
R.05.03.06	Party Y canceled a reservation as requested by Party X	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse set to SUCCESS and the `error` and `payload` fields of the AsyncResponse both left unset.

NOTE	An unused and canceled Reservation of an EVSE may result in cost being incurred by Party X, thus also in a Session and a CDR being replicated from Party Y to Party X.

5.3.3. UC: 05.04 - Cancel a Reservation as a CPO

1	**Objective(s)**	**1.** Party Y notifies Party X that Party Y is not capable of fulfilling a reservation made by Party X
2	**Description**	Sometimes a CPO is unfortunately forced to cancel a reservation that it previously accepted. By making a request to the asynchronous response URL of the reservation request it can do so.
3	**Actors**	CPO, eMSP
4	**Flow**	1. Party Y makes a request to the asynchrous response URL of Party X's reservation remote procedure call with a "FAILED" result type.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Locations to Party X on Platform A. Party X on Platform A made a reservation of an EVSE on a Location issued by Party Y on Platform B. This reservation has not yet expired.
6	**Postconditions**	Party X knows that Party Y will not be able to fulfill the reservation.
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	This functionality of OCPI is not to be used lightly. When a driver makes a reservation of an EVSE, they want to be sure to have a charging location. So if the CPO cancels the reservation, the driver will for sure not like it. There are some circumstances however where the CPO is forced to cancel a reservation. For example, consider the situation that the Charging Station has broken down, or the CPO is notified of ongoing roadworks which makes the Charging Station unreachable, etc.

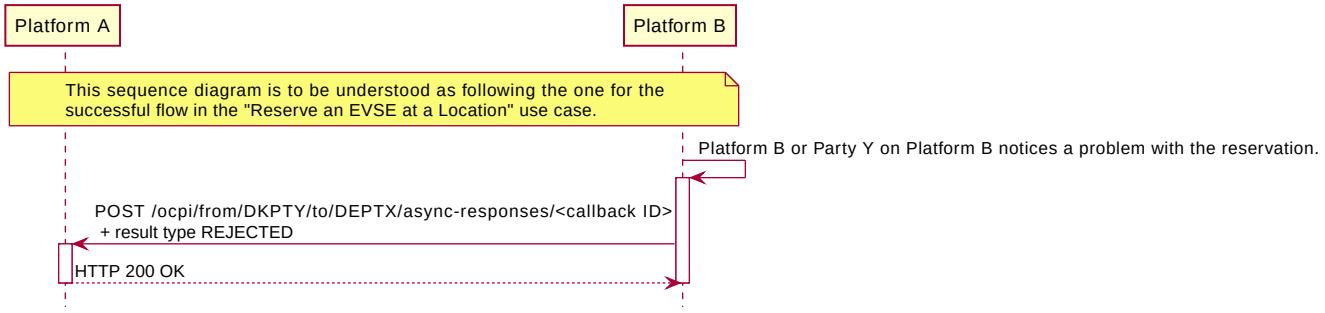


Figure 33. Sequence Diagram: Cancel a Reservation as a CPO

Table 29. UC: 05.04 Requirements

ID	Precondition	Requirement
R.05.04.01		Platform B SHALL make a request according to Platform A according to Make a request on behalf of a Party to a Party on another Platform with Party Y as the sender and Party X as the receiver.
R.05.04.02		Platform B SHALL use the POST request verb for its request.
R.05.04.03		Platform B SHALL use the same relative path for its request that it used before when delivering the asynchronous response when the reservation was made according to Reserve an EVSE at a Location.
R.05.04.04		Platform B SHALL set the result_type field of the AsyncResponse object in the body of its request to FAILED.
R.05.04.05		Platform B MAY set the payload field of the AsyncResponse object in the body of its request to a string describing why it has to cancel the reservation
R.05.04.06		Platform B MAY leave the payload field of the AsyncResponse object in the body of its request unset
R.05.04.07	Platform A received Platform B's request according to requirements R.05.04.01 - R.05.04.06.	Platform A SHALL respond with an OcpiResponse object in the response body that leaves the `data` field unset and that has the `status_code` field set to 1000.

5.3.4. UC: 05.06 - Unlock a Connector

1	**Objective(s)**	**1.** Party Y unlocks a Connector at which a Driver related to Party X is charging.
2	**Description**	The Unlock Connector functionality 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 Charging Station tries to unlock the connector. In such a situation the EV-Driver can contact either the CPO or the eMSP to retry the unlocking. If they contact the eMSP, the eMSP has no direct access to the Charging Station. The eMSP may want to use OCPI to request a Connector unlock from the CPO. To do so, the eMSP (Party X) makes an asynchronous remote procedure call to the CPO (Party Y). In the request Party X sends the identifiers of the Connector that they wish to unlock. In the response Party Y sends a confirmation or a rejection of the unlock request.
3	**Actors**	eMSP, CPO

4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains identifiers to identify the Connector that is to be unlocked. 2. Party Y determines if it is willing to execute the requested unlock operation. 3. Platform B responds to Platform A on behalf of Party Y to inform Party X if their request was accepted. 4. If the request was accepted, Party Y proceeds to unlock the connector. 5. If the request was accepted, Platform B sends a request to Platform A on behalf of Party Y to inform Party X of the result of the unlock operation.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Locations to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to unlock a Connector at one of Party Y's Locations. * Party Y attempted to unlock a Connector at Party X's request. Also Party X knows whether this attempt was successful or not.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party. If Party Y received the request and unlocked the Connector for Party X, but Platform B failed to deliver an asynchronous response to Platform A for Party X, then the connection was unlocked but Party X does not know about it. There is no automatic remediation process for this condition.
8	**Remark(s)**	The Unlock Connector functionality is designed to be used by operator staff at e.g. an MSP. It is not intended for direct use by charging app end users as specified in R.05.06.06.

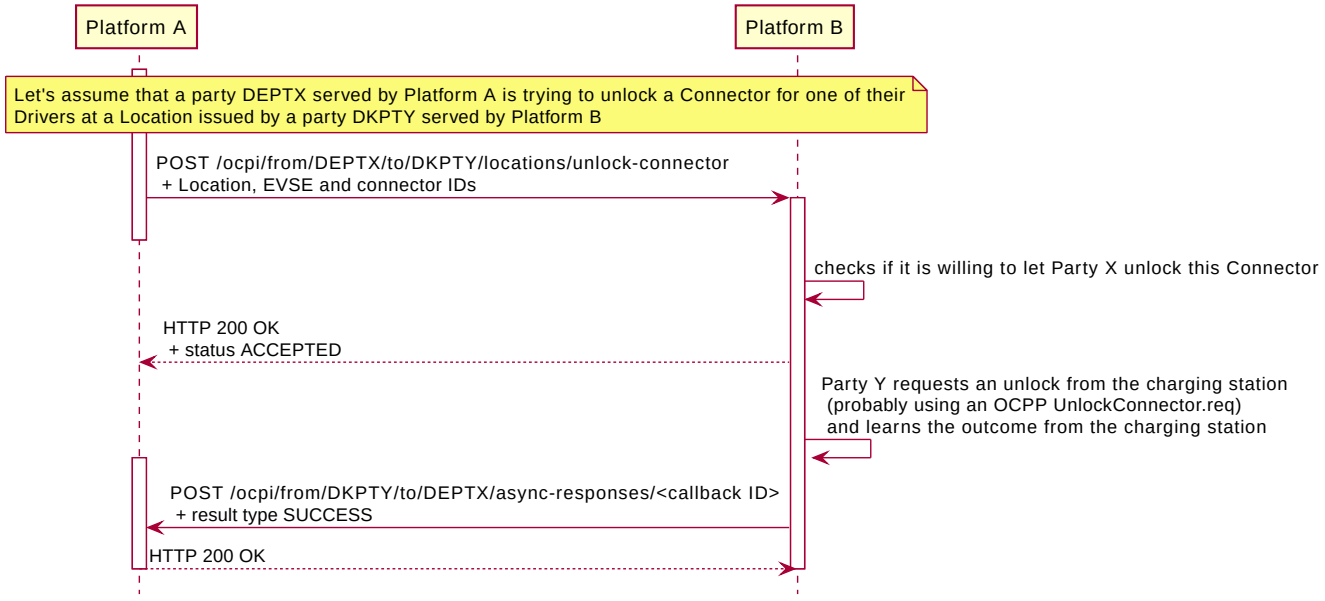


Figure 34. Sequence Diagram: Unlock a Connector

Table 30. UC: 05.06 Requirements

ID	Precondition	Requirement
R.05.06.01		Platform A SHALL make the request to unlock a Connector following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.06.02		Platform A SHALL use "unlock-connector" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.06.03		Platform A SHALL use a UnlockConnectorRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.06.04	Party Y cannot or will not act upon the unlock request because of the status of the EVSE or the Connector or the device containing the EVSE and the Connector	Platform B SHALL respond to Platform A's request with a response with ACCEPTED in the data field of the OcpiResponse object. Platform B SHALL then send a request with an AsyncResponse object with the payload field unset and with the error field set to an appropriate ChargingStationCommandError object.
R.05.06.05	Party Y received a confirmation from the device that it accepted the reset request.	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse object set to SUCCESS, and the payload and error fields of this AsyncResponse left unset.
R.05.06.06		Party X and Platform A SHALL NOT provide Drivers with the opportunity to trigger the process described in this use case without review by staff working for Party X.

5.3.5. UC: 05.07 - Reset an EVSE

1	**Objective(s)**	**1.** Party Y resets an EVSE at which a Driver related to Party X is charging.
2	**Description**	The Reset EVSE functionality is intended to be used in the rare situation that an EVSE is not correctly responding to a Driver's interactions. Many such problem situations are caused by the hardware of software ending up in an unanticipated state. These problems are typically resolved by requesting the charging station device to perform a reset. This use case gives the eMSP the opportunity to send reset requests to charging devices without involving CPO staff. Note that it is optional for CPOs to offer this access to their hardware to eMSPs; they may simply reject all reset requests from eMSPs if they would rather not open up their devices to reset requests from third parties. They can indicate this unwillingness to reset their hardware at eMSPs' requests by not supplying the RESET_CAPABLE capability in the ChargingStation objects in their Locations. To request a reset from a device, the eMSP (Party X) makes an asynchronous remote procedure call to the CPO (Party Y). In the request Party X sends the identifiers of the EVSE for which they wish to reset the device. In the response Party Y sends a confirmation or a rejection of the reset request.
3	**Actors**	eMSP, CPO

4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains identifiers to identify the EVSE for which the device is to be reset. 2. Party Y determines if it is willing to execute the requested reset operation. 3. Platform B responds to Platform A on behalf of Party Y to inform Party X if their request was accepted. 4. If the request was accepted, Party Y proceeds to reset the device. 5. If the request was accepted, Platform B sends a request to Platform A on behalf of Party Y to inform Party X of the result of the reset operation.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Locations to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to reset a device at one of Party Y's Locations. * Party Y attempted to reset a device at Party X's request. Also Party X knows whether this attempt was successful or not.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party. If Party Y received the request and reset the device for Party X, but Platform B failed to deliver an asynchronous response to Platform A for Party X, then the device was reset but Party X does not know about it. There is no automatic remediation process for this condition.

8	**Remark(s)**	The Reset EVSE functionality is designed to be used by operator staff at e.g. an MSP. It is not intended for direct use by charging app end users as specified in R.05.07.06. OCPI does not concern itself with how EVSEs are grouped according to their connection to a back-office system or their control electronics. As a result, this use case cannot and does not specify exactly how a CPO decides which device is to be reset based on the Location ID and EVSE UID given in the reset request. It is up to the CPO to decide on a way to determine the right device to reset, or alternatively, to not offer this functionality to eMSPs. Whichever way a CPO chooses, they will typically also want to take into account that a reset to help one Driver should not interfere with the service of other Drivers. For example, it is probably not the right course of action for a CPO to do a full "hard" reset of the central control unit of a fifty EVSE charging facility because an eMSP requested a reset for one EVSE. Typically, a CPO that supports the Reset EVSE functionally will want to have a policy in place that rejects Reset EVSE requests by default. Only under specific circumstances should Reset EVSE requests typically be acted upon by the CPO. Such circumstances might be that the request comes from a Party who is known to be a customer support provider to the CPO, or that the request comes from an eMSP who issued the Charging Token that is used for a currently ongoing Charging Session at the EVSE. As noted under there, the RESET_CAPABLE capability being present on a Charging Station does not create an obligation on the issuer of the Location to act upon any Reset EVSE requests.

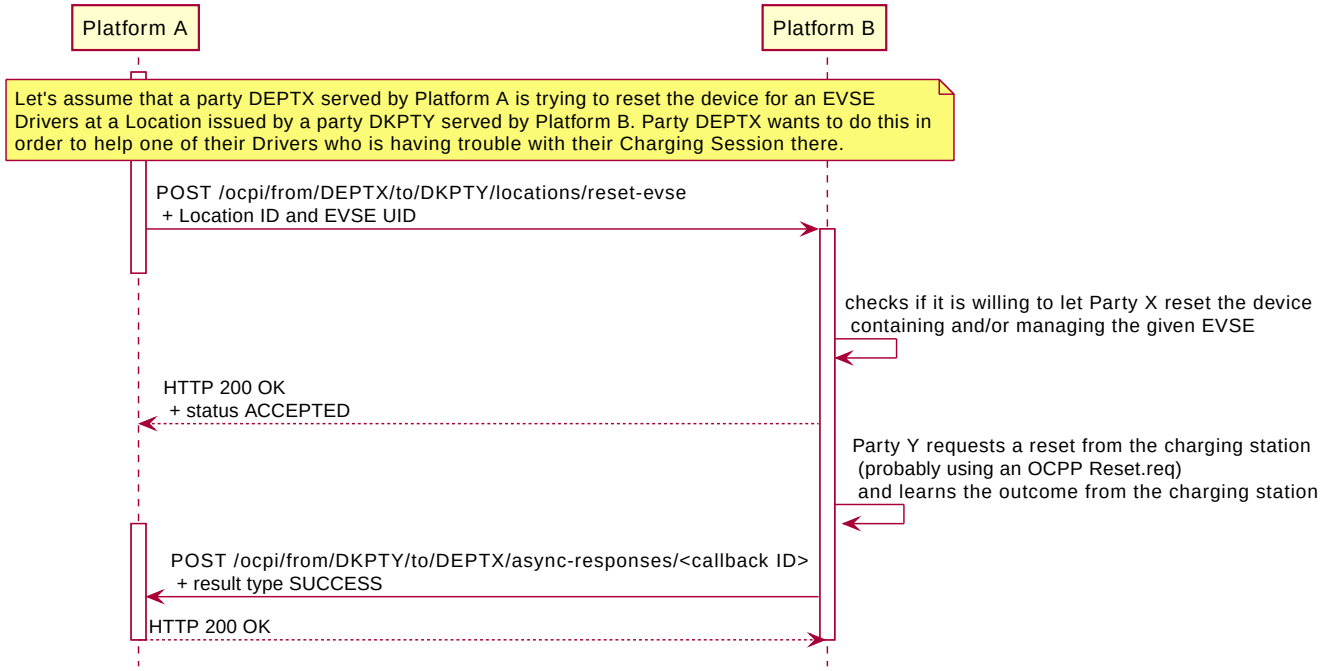


Figure 35. Sequence Diagram: Reset an EVSE

Table 31. UC: 05.07 Requirements

ID	Precondition	Requirement
R.05.07.01		Platform A SHALL make the request to reset the device for an EVSE following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.07.02		Platform A SHALL use "reset-evse" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.07.03		Platform A SHALL use a ResetEvseRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.05.07.04	Party Y cannot or will not act upon the reset request because of the status of the EVSE or the Connector or the device containing the EVSE and the Connector	Platform B SHALL respond to Platform A's request with a response with ACCEPTED in the data field of the OcpiResponse object. Platform B SHALL then send a request with an AsyncResponse object with the payload field unset and with the error field set to an appropriate ChargingStationCommandError object.
R.05.07.05	Party Y unlocked the Connector	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse object set to SUCCESS, and the payload and error fields of this AsyncResponse left unset.
R.05.07.06		Party X and Platform A SHALL NOT provide Drivers with the opportunity to trigger the process described in this use case without review by staff working for Party X.

5.4. Object type definitions

5.4.1. AdditionalGeoLocation *class*

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

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

5.4.2. Address *class*

This gives information on how to locate the Location.

Property	Type	Card.	Description
address	UnicodeString[1..45]	1	Street/block name and house number if available.
city	UnicodeString[1..45]	1	City or town.
postal_code	UnicodeString[1..10]	?	Postal code of the Location.
state	UnicodeString[1..20]	?	State or province of the Location. This is intended to be used only in locales where a state or province is commonly given in addresses. This field would typically be filled for Locations in the United States of America and be left unset for Locations in The Netherlands for example.
country	CiAsciiString[3]	1	ISO 3166-1 alpha-3 code for the country of this Location.
coordinates	GeoLocation	1	Coordinates of the Location. This could be the geographical location of one or more Charging Stations within a facility, but it can also be the entrance of a parking or other facility where Charging Stations are located. It is up to the CPO to use the point that makes the most sense to a Driver for a given Location. Once arrived at the Location's coordinates, any further instructions to reach a Charging Station from the Location coordinates are stored in the Charging Station object itself (such as the floor number, visual identification or written instructions).

5.4.3. CancelReservationRequest *class*

With CancelReservation the Sender can request the Cancel of an existing Reservation. The CancelReservation needs to contain the reservation_id that was given by the Sender to the ReserveNow.

Property	Type	Card.	Description
reservation_id	CiAsciiString[1..36]	1	The ID of the reservation that cancelation is requested for.

5.4.4. Capability *enum*

The capabilities of a Charging Station.

Value	Description
CHARGING_PROFILE_CAPABLE	The Charging Station supports charging profiles.

Value	Description
CHARGING_PREFERENCES_CAPABLE	The Charging Station supports charging preferences.
CHIP_CARD_SUPPORT	The Charging Station has a payment terminal that supports chip cards.
CONTACTLESS_CARD_SUPPORT	The Charging Station has a payment terminal that supports contactless cards.
CREDIT_CARD_PAYABLE	The Charging Station has a payment terminal that makes it possible to pay for charging using a credit card.
DEBIT_CARD_PAYABLE	The Charging Station has a payment terminal that makes it possible to pay for charging using a debit card.
PED_TERMINAL	The Charging Station has a payment terminal with a pin-code entry device.
REMOTE_START_CAPABLE	Sessions on the Charging Station can be started through OCPI RPC use case Start a Session.
REMOTE_STOP_CAPABLE	Sessions on the Charging Station can be stopped through OCPI RPC use case Stop a Session.
RESERVABLE	The Charging Station's EVSEs can be reserved.
RESET_CAPABLE	The Party that issued the Location with this Charging Station can accept requests for remote resets of the device containing this Charging Station as specified in use case Reset an EVSE. The RESET_CAPABLE capability does not create an obligation on the issuer of the Location to accept any request for a reset. The RESET_CAPABLE capability merely signals that a reset request may be accepted by the CPO under some conditions. These conditions may not be transparent to the Party receiving the Location. Also note that OCPP uses a different definition of Charging Station from the one used by OCPI and other standards. Therefore there is no guarantee that the effect of a Reset on a certain EVSE is limited to the EVSEs that are in the same OCPI Charging Station.
RFID_READER	Charging at this Charging Station can be authorized with an RFID token.
START_SESSION_CONNECTOR_REQUIRED	When a StartSessionRequest is sent to an EVSE of Charging Station, the MSP is required to add the optional `connector_id` field in the StartSessionRequest object.
TOKEN_GROUP_CAPABLE	This Charging Station supports token groups, two or more tokens work as one, so that a session can be started with one token and stopped with another (handy when a card and key-fob are given to the EV-driver).
UNLOCK_CAPABLE	Connectors on this Charging Station have mechanical lock that can be requested by the eMSP to be unlocked.

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

There are Charging Stations in the field that still use OCPP 1.6 or older OCPP versions. If these Charging Station have multiple connectors per EVSE, the CPO needs to know which connector to start when receiving a StartSessionRequest for a session on such a Charging Station. If this is the case, the CPO should set the `START_SESSION_CONNECTOR_REQUIRED` capability on such a Charging Station.

5.4.5. ChargingStation *class*

The ChargingStation object describes a device that is physically distinct and has a single user interface. It always belongs to a Location object. The object only contains directions to get from the central Location entrance to the Charging Station (i.e. floor, physical_reference or directions).

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

A ChargingStation object contains a list with the Charging Station's EVSEs.

Note that OCPI's definition of "Charging Station" is the same one that is used by EMI3 and EU regulations, but different from the one used by OCPP as of OCPP versions 1.5, 1.6 and 2.0.1. In a situation where a single OCPP connection is used to connect multiple physically distinct booths or poles that each have their own screens and/or RFID readers, there are multiple Charging Stations for OCPI, while there is only one OCPP "Charging Station".

Property	Type	Card.	Description
id	CiAsciiString[1..36]	1	An identifier that uniquely indentifiers this Charging Station among all Charging Stations in all Locations issued by the same Party.
evses	EVSE	*	List of EVSEs that belong to this Charging Station.
capabilities	Capability	*	List of functionalities that the Charging Station is capable of.
floor_level	AsciiString[1..4]	?	Level on which the Charging Station is located (in garage buildings) in the locally displayed numbering scheme.
coordinates	GeoLocation	?	Coordinates of the Charging Station.
physical_reference	UnicodeString[1..16]	?	A number/string printed on the outside of the Charging Station for visual identification.
directions	DisplayText	*	Multi-language human-readable directions when more detailed information on how to reach the Charging Station from the *Location* is required.
images	Image	*	Links to images related to the Charging Station such as photos or logos.
last_updated	DateTime	1	Timestamp when this Charging Station or one of its EVSEs was last updated (or created).

5.4.6. ChargingStationCommandError *class*

Property	Type	Card.	Description
status	ChargingStationCommandStatus	1	An error code that signals why the requested operation was not executed or failed

Property	Type	Card.	Description
message	DisplayText	*	Human-readable description of the reason for the status (if one can be provided), multiple languages can be provided.

5.4.7. ChargingStationCommandStatus *enum*

Value	Description
DEVICE_OFFLINE	The operation could not be performed because remote communication is not available to the device that has to execute the operation.
EVSE_OCCUPIED	The operation could not be performed because the EVSE that the operation was requested on is occupied.
EVSE_INOPERATIVE	The operation could not be performed because the EVSE that the operation was requested on is inoperative.

5.4.8. Connector *class*

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

Property	Type	Card.	Description
id	CiAsciiString[1..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.
cable_length	number	?	The length of the attached cable in centimeters. Only applicable if the value of the `format` field is `CABLE`.
power_type	PowerType	1	
max_voltage	int	1	Maximum voltage of the connector (line to neutral for AC_3_PHASE), in volt [V]. For example: DC Chargers might vary the voltage during charging when battery almost full.
max_amperage	int	1	Maximum amperage of the connector, in ampere [A].

Property	Type	Card.	Description
max_electric_power	int	?	Maximum electric power that can be delivered by this connector, in Watts (W). When the maximum electric power is lower than the calculated value from `voltage` and `amperage`, this value should be set. For example: A DC Charging Station which can delivers up to 920V and up to 400A can be limited to a maximum of 150kW (max_electric_power = 150000). Depending on the car, it may supply max voltage or current, but not both at the same time. For AC Charging Stations, the amount of phases used can also have influence on the maximum power.
terms_and_conditions	URL	?	URL to the operator's terms and conditions.
capabilities	ConnectorCapability	*	A list of functionalities that the connector is capable of.
last_updated	DateTime	1	Timestamp when this Connector was last updated (or created).

5.4.9. ConnectorCapability *OpenEnum*

Functionalities that a Connector may or may not support.

Note that these capabilities are meant to signal to eMSPs and their Drivers that a Driver can indeed use these functionalities at a Connector. Mere support for a standard by the charging hardware is not enough to warrant the presence of these capabilities.

Value	Description
IEC_15118_2_PLUG_AND_CHARGE	The Connector supports authentication of the Driver using a contract certificate stored in the vehicle according to IEC 15118-2.
IEC_15118_20_PLUG_AND_CHARGE	The Connector supports authentication of the Driver using a contract certificate stored in the vehicle according to IEC 15118-20.

NOTE ConnectorCapability is an OpenEnum while Capability is not. This is done because ConnectorCapability serves to signal which communication standards between vehicle and EVSE are supported, and it is easy to imagine more such standards appearing during the lifecycle of OCPI 3.0. Capability on the other hand is about less clearly defined human-directed services and encouraging Parties to add their own values there would easily lead to oversharing of non-standardized services.

5.4.10. ConnectorFormat *enum*

The format of the Connector, that is, 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.

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

5.4.11. ConnectorType *OpenEnum*

The socket or plug standard of the charging point.

For convenience the possible values are grouped in three separate tables by general plug category: common EV charging connectors, household/industrial connectors, and rare legacy or novelty connectors.

5.4.12. Common EV Charging Connector Types

These are the EV specific connectors typically used on publicly accessible charging stations as of 2024.

Value	Description
CHADEMO	CHAdeMO
IEC_62196_T1	IEC 62196 Type 1 "SAE J1772"
IEC_62196_T1_COMBO	Combo Type 1 based, "CCS1"
IEC_62196_T2	IEC 62196 Type 2 "Mennekes"
IEC_62196_T2_COMBO	Combo Type 2 based, "CCS2"
MCS	The MegaWatt Charging System (MCS) connector as developed by CharIN
SAE_J3400	SAE J3400, also known as North American Charging Standard (NACS), developed by Tesla, Inc in 2021.
TESLA_S	Tesla Connector "Model-S"-type (oval, 5 pin). Mechanically compatible with SAE J3400 but uses CAN bus for communication instead of power line communication.

5.4.13. Domestic and Industrial Connector Types

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 pins
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_A	**Standard/Domestic household, type "A", NEMA 1-15, 2 pins**
DOMESTIC_L	Standard/Domestic household, type "L", CEI 23-16-VII, 3 pins
DOMESTIC_M	Standard/Domestic household, type "M", BS 546, 3 pins
DOMESTIC_N	Standard/Domestic household, type "N", NBR 14136, 3 pins
DOMESTIC_O	Standard/Domestic household, type "O", TIS 166-2549, 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 phases 16 amperes (usually red)
IEC_60309_2_three_32	IEC 60309-2 Industrial Connector three phases 32 amperes (usually red)
IEC_60309_2_three_64	IEC 60309-2 Industrial Connector three phases 64 amperes (usually red)
NEMA_5_20	NEMA 5-20, 3 pins
NEMA_6_30	NEMA 6-30, 3 pins
NEMA_6_50	NEMA 6-50, 3 pins
NEMA_10_30	NEMA 10-30, 3 pins
NEMA_10_50	NEMA 10-50, 3 pins
NEMA_14_30	NEMA 14-30, 3 pins, rating of 30 A
NEMA_14_50	NEMA 14-50, 3 pins, rating of 50 A

5.4.14. Legacy and Novelty Connector Types

CHAOJI	**The ChaoJi connector. The new generation charging connector, harmonized between CHAdeMO and GB/T. DC.**
GBT_AC	Guobiao GB/T 20234.2 AC socket/connector
GBT_DC	Guobiao GB/T 20234.3 DC connector
IEC_62196_T3A	IEC 62196 Type 3A
IEC_62196_T3C	IEC 62196 Type 3C "Scame"
PANTOGRAPH_BOTTOM_UP	On-board Bottom-up-Pantograph typically for bus charging
PANTOGRAPH_TOP_DOWN	Off-board Top-down-Pantograph typically for bus charging
TESLA_R	Tesla Connector "Roadster"-type (round, 4 pin)

5.4.15. 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.
environ_impact	EnvironmentalImpact	*	Key-value pairs (enum + percentage) of nuclear waste and CO2 exhaust of this Location.
supplier_name	UnicodeString[1..64]	?	Name of the energy supplier, delivering the energy for this Location.*
energy_product_name	UnicodeString[1..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.)*

5.4.15.1. Examples

Simple:

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

5.4.16. EnergySource *class*

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

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

5.4.17. 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.

Value	Description
SOLAR	Regenerative power from PV.
WIND	Regenerative power from wind turbines.
WATER	Regenerative power from water turbines.

5.4.18. EnvironmentalImpact *class*

Amount of waste produced/emitted per kWh.

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

5.4.19. EnvironmentalImpactCategory *enum*

Categories of environmental impact values.

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

5.4.20. EVSE *class*

The EVSE object describes the part that controls the power supply to a single EV in a single session. It always belongs to a ChargingStation object.

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.

Field Name	Type	Cardinality	Description
uid	CiAsciiString[1..36]	1	Uniquely identifies the EVSE among all EVSEs of all Locations of the same Party. This field can never be changed, modified or renamed. This is the 'technical' identification of the EVSE, not to be used as 'human readable' identification, use the field `evse_id` for that. This field is named `uid` instead of `id`, because `id` could be confused with `evse_id` which is the field containing an ID in the EMI3 defined "EVSE-ID" format. Note that in order to fulfill both the requirement that an EVSE's uid be unique within a CPO's platform and the requirement that EVSEs are never deleted, a CPO will typically want to avoid using identifiers of the physical hardware for this uid property. If they do use such a physical identifier, they will find themselves breaking the uniqueness requirement for uid when the same physical EVSE is redeployed at another Location.
evse_id	CiAsciiString[1..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 in the real world, the `evse_id` can be removed from an EVSE object if the `status` is set to `REMOVED`.
presence	PresenceStatus	1	Whether this EVSE is currently physically present, or only planned for the future, or already removed.
status_schedule	StatusSchedule	*	Indicates a planned status update of the EVSE.
connectors	Connector	+	List of available connectors on the EVSE.
physical_reference	UnicodeString[1..16]	?	A number/string printed on the outside of the EVSE for visual identification.
parking	Parking	1	A description of the available parking for the EVSE.
images	Image	*	Links to images related to the EVSE such as photos or logos.
calibration_info_url	URL	?	Link to a URL where certificates, identifiers and public keys related to the calibration of meters in this EVSE can be found.
last_updated	DateTime	1	Timestamp when this EVSE or one of its Connectors was last updated (or created).

5.4.21. EvsePosition *enum*

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

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

For EVSEs located near the corner of a parking bay, LEFT or RIGHT should be used as appropriate according to the side of the vehicle the corner is on.

5.4.22. ExceptionalPeriod *class*

Specifies one exceptional period for opening or access hours.

Property	Type	Card.	Description
period_begin	DateTime	1	Begin of the exception. The timestamp is given in UTC as all DateTime fields. The Location's `time_zone` field can be used to convert to local time.
period_end	DateTime	1	End of the exception. The timestamp is given in UTC as all DateTime fields. The Location's `time_zone` field can be used to convert to local time.

5.4.23. 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.

Value	Description
NATURE	Located in, or close to, a park, nature reserve etc.
MUSEUM	A museum.
BIKE_SHARING	A bike/e-bike/e-scooter sharing location.
BUS_STOP	A bus stop.
TAXI_STAND	A taxi stand.
TRAM_STOP	A tram stop/station.
METRO_STATION	A metro station.
TRAIN_STATION	A train station.
AIRPORT	An airport.
PARKING_LOT	A parking lot.
CARPOOL_PARKING	A carpool parking.
FUEL_STATION	A Fuel station.

5.4.24. GeoLocation *class*

This class defines the geographical location of the Charging Station. The geodetic system to be used is WGS 84.

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

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

5.4.25. Hours *class*

Opening and access hours of the Location.

Property	Type	Card.	Description
twentyfourseven	boolean	1	True to represent 24 hours a day and 7 days a week, except the given exceptions.

Property	Type	Card.	Description
regular_hours	RegularHours	*	Regular hours, weekday-based. Only to be used if `twentyfourseven=false`, then this field needs to contain at least one RegularHours object. The time windows given in the different RegularHours objects may not overlap. The RegularHours objects must be ordered ascendingly, first by the `weekday` field and then by the `period_begin` field.
exceptional_openings	ExceptionalPeriod	*	Exceptions for specified calendar dates, time-range based. Periods the station is operating/accessible. Additional to `regular_hours`. May overlap regular rules.
exceptional_closings	ExceptionalPeriod	*	Exceptions for specified calendar dates, time-range based. Periods the station is not operating/accessible. Overwriting `regular_hours` and `exceptional_openings`. Should not overlap `exceptional_openings`.

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

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

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

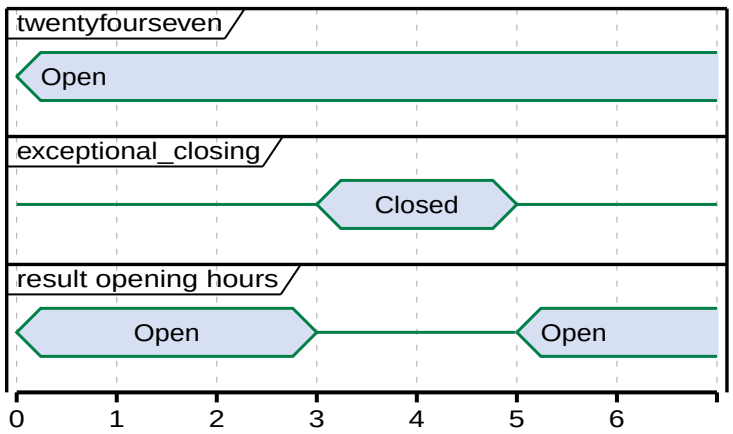


Figure 36. Diagram showing a representation of the example 24/7 open with exception closing.

5.4.25.2. Example: Opening Hours with exceptional closing.

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

```
{
  "twentyfourseven": false,
  "regular_hours": [{
```

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

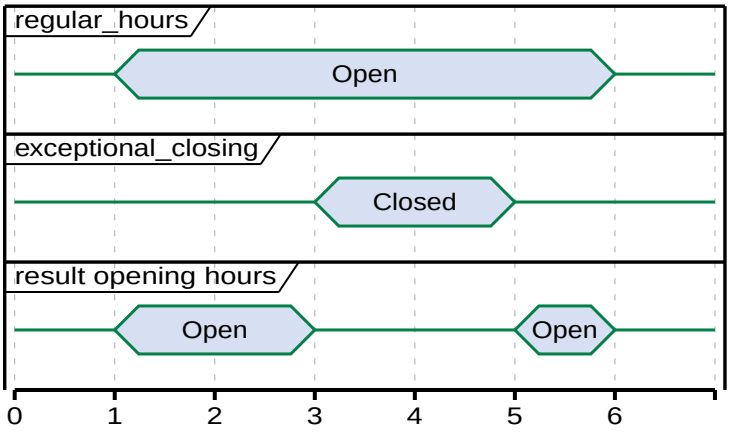


Figure 37. Diagram showing a representation of the example Opening Hours with exceptional closing

5.4.25.3. Example: Opening Hours with exceptional opening.

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

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

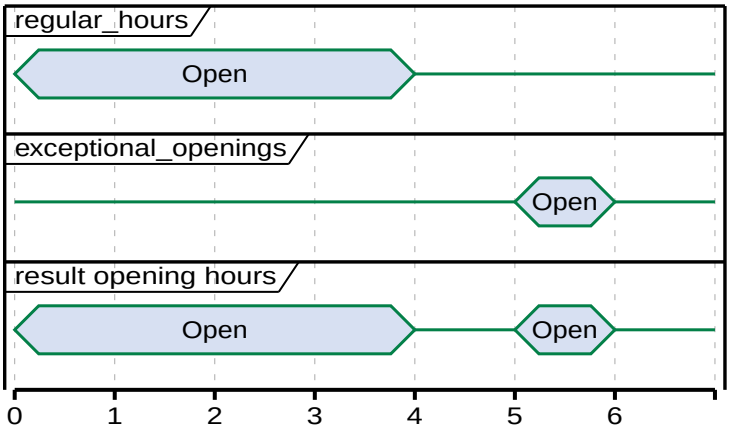


Figure 38. Diagram showing a representation of the example Opening Hours with exceptional opening.

5.4.26. Parking *class*

Describes the restrictions on parking that is available for an EVSE.

This object describes the available parking. It may not describe an identifiable physical parking bay. The reason is that for some EVSEs, no identifiable delineated parking bays are available. This occurs a lot with streetside parking for example.

Property	Type	Card.	Description
vehicle_types	VehicleType	+	The vehicle types that the EVSE is intended for and that the associated parking is designed to accomodate.
max_vehicle_weight	number	?	The maximum vehicle weight that can park at the EVSE, in kilograms. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
max_vehicle_height	number	?	The maximum vehicle height that can park at the EVSE, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
max_vehicle_length	number	?	The maximum vehicle length that can park at the EVSE, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
max_vehicle_width	number	?	The maximum vehicle width that can park at the EVSE, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
parking_bay_length	number	?	The length of the parking bay, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.

Property	Type	Card.	Description
parking_bay_width	number	?	The width of the parking bay, in centimeters. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
dangerous_goods_allowed	boolean	?	Whether vehicles loaded with dangerous substances are allowed to park at the EVSE. A value for this field should be provided unless the value of the `vehicle_types` field contains no values other than `PERSONAL_VEHICLE` or `MOTORCYCLE`.
evse_position	EvsePosition	1	The position of the EVSE relative to the parking space.
direction	ParkingDirection	1	The direction in which the vehicle is to be parked next to this EVSE.
restricted_to_type	boolean	1	Whether it is forbidden for vehicles of a type not listed in `vehicle_types` to park at this EVSE, even if they can physically park there safely.
parking_restrictions	ParkingRestriction	*	All applicable restrictions on who can park at this EVSE, apart from those related to the vehicle type.
reservation_required	boolean	1	Whether a reservation is required for parking at the EVSE.
time_limit	number	?	A time limit. If this field is present, vehicles may not park in this parking longer than this number of minutes.
roofed	boolean	?	Whether the vehicle will be parked under a roof while charging.
images	Image	*	Photos of the parking space at the EVSE. At least one photograph should be provided if the value of `vehicle_types` includes the `DISABLED` vehicle type.
lighting	boolean	?	Whether the parking space for the EVSE is lit by artificial lighting.
standards	UnicodeString[1..36]	*	A list of standards that the parking space conforms to, e.g. PAS 1899 for parking for people with disabilities.

NOTE The reason that there are separate fields for the maximum vehicle dimensions and the parking bay dimensions is that these fields can indeed have different values for parking bays for people with disabilities.

5.4.27. ParkingDirection *enum*

Value	Description
PARALLEL	Parking happens parallel to the roadway on which vehicles approach the EVSE.
PERPENDICULAR	Parking happens perpendicular to the roadway on which vehicles approach the EVSE.

Value	Description
ANGLE	Parking happens at an angle to the roadway on which vehicles approach the EVSE (i.e. echelon parking).
DRIVE_THROUGH	A vehicle can stop, charge, and proceed without reversing into or out of a parking bay.

5.4.28. ParkingRestriction *class*

A restriction on which groups of drivers can use an EVSE's parking and thereby the EVSE itself.

Property	Type	Card.	Description
group	ParkingRestrictionGroup	+	One or more groups that drivers have to be in to be allowed to park here.
applies_outside_opening_hours	boolean	1	Whether the restriction applies also outside opening hours of the establishment that the Location belongs to. This field can be used for example to create a ParkingRestriction that signals that a certain EVSE is for employees only during opening hours, but can be used by everyone outside those opening hours.

5.4.29. ParkingRestrictionGroup *OpenEnum*

An enumeration of possible groups that parking may be restricted to.

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

OCPI implementers are not required or otherwise expected to make their software check if these restrictions are fulfilled. These restrictions are typically shown to Drivers in driver apps and enforced by on-site staff.

5.4.30. ParkingType *enum*

Reflects the general type of the Charging Station's location. This may be used for Driver information.

Value	Description
ALONG_MOTORWAY	Location on a parking facility/rest area along a motorway, freeway, interstate, highway etc.

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

5.4.31. Location *class*

Property	Type	Card.	Description
publish	boolean	1	Whether the receiving Party or Platform may publish the Location. When this is set to `false`, the receiving Party or Platform MAY NOT disclose information from this Location object to anyone not holding a Token listed in the field `publish_allowed_to`. When the same physical facility has both some EVSEs that may be published and other ones that may not be published, the sender Party SHOULD send two separate Location objects for the two groups of EVSEs.
publish_allowed_to	PublishTokenType	*	This field SHALL NOT be used unless the `publish` field is set to `false`. Only holders of Tokens that match all the set fields of one PublishToken in the list are allowed to be shown this Location.
name	UnicodeString[1..255]	?	Display name of the Location.
address	Address	?	Address and geographical location of the Location. This has to be present unless the `publish` field is set to false.
related_locations	AdditionalGeoLocation	*	Geographical location of related points relevant to the user.
parking_type	ParkingType	?	The general type of parking at the Location.
charging_pool	ChargingStation	+	The Charging Pool of this Location, that is, the list of Charging Stations that make up the physical charging infrastructure of 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 with Use Case Request Parties Served by Platform, selected by the Party ID of the Party that issued this Location, MAY be used instead.

Property	Type	Card.	Description
suboperator	BusinessDetails	?	Information of the suboperator if available.
owner	BusinessDetails	?	Information of the owner if available.
services	LocationService	*	Optional list of services that are offered at the Location by the CPO or their affiliated partners.
facilities	Facility	*	Optional list of facilities that this Location directly belongs to.
time_zone	AsciiString[1..255]	1	One of the TZ-values from [TZVAL] representing the time zone of the Location. Examples: "Europe/Oslo", "Europe/Zurich".
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. That is, when the parking garage closes its barriers over night, can vehicles 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.
max_power	LocationMaxPower	?	How much power or current this Location can draw from the grid at any one time.
help_phone	CiAsciiString[1..25]	?	A telephone number that a Driver using the Location may call for assistance. Calling this number will typically connect the caller to the CPO's customer service department.
last_updated	DateTime	1	Timestamp when this Location or one of its EVSEs or Connectors were last updated (or created).

5.4.32. LocationMaxPower *class*

Property	Type	Card.	Description
unit	ChargingRateUnit	1	The unit in which the maximum draw is expressed.
value	number	1	The maximum power or current that the Location can draw.

5.4.33. PowerType *enum*

Value	Description
AC_1_PHASE	AC single phase.
AC_2_PHASE	AC two phases, only two of the three available phases connected.

Value	Description
AC_2_PHASE_SPLIT	AC two phases using split phase system.
AC_3_PHASE	AC three phases.
DC	Direct Current.

5.4.34. PresenceStatus *enum*

Whether an EVSE is currently present.

Value	Description
PRESENT	The EVSE is currently present and whether it is currently usable can be indicated using the EVSE Status module.
PLANNED	The EVSE is not currently present but it is planned for the future.
REMOVED	The EVSE is not currently present but it is used to be present in the past.

5.4.35. PublishTokenType *class*

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

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

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

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

Property	Type	Card.	Description
uid	CiAsciiString[1..36]	?	Unique ID by which this Token can be identified.
type	TokenType	?	Type of the token.
visual_number	UnicodeString[1..64]	?	Visual readable number/identification as printed on the Token (RFID card).
issuer	UnicodeString[1..64]	?	Issuing company, most of the times the name of the company printed on the token (RFID card), not necessarily the eMSP.
group_id	CiAsciiString[1..36]	?	This ID groups a couple of tokens. This can be used to make two or more tokens work as one.

5.4.36. RegularHours *class*

Regular recurring operation or access hours.

Property	Type	Card.	Description
weekday	int[1]	1	Number of day in the week, from Monday (1) till Sunday (7)

Property	Type	Card.	Description
period_begin	AsciiString[5]	1	Begin of the regular period, in local time, given in hours and minutes. Must be in 24h format with leading zeros. Example: "18:15". Hour/Minute separator: ":" Regex: `([0-1][0-9]|2[0-3]):[0-5][0-9]`. In this field, "00:00" means midnight at the beginning of the day.
period_end	AsciiString[5]	1	End of the regular period, in local time, syntax as for `period_begin`. Must be later than `period_begin` or be "00:00" to signal that the charging station is open until midnight at the end of the day.

5.4.36.1. Handling midnight

The special meanings of the "00:00" value make that `{"weekday": 6, "period_begin": "00:00", "period_end": "00:00"}` means that a Location is opened all 24 hours of Saturday.

The values of the `weekday` field are not required to be unique among the RegularHours objects in the `regular_hours` field of the Hours object. This makes that an opening time window that stretches across midnight can be given by including two entries in the `regular_hours` field of the Hours object like this:

```
...
"hours": [
    {"weekday": 6, "period_begin": "12:00", "period_end": "00:00"},
    {"weekday": 7, "period_begin": "00:00", "period_end": "02:00"},
    {"weekday": 7, "period_begin": "12:00", "period_end": "22:00"}
  ],
...
```

In the example above, the Location is open on Saturday from noon until 2 AM, and on Sunday from noon until 10 PM.

5.4.36.2. Example with exceptional opening hours

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

```
  "opening_times": {
    "regular_hours": [
      {
        "weekday": 1,
        "period_begin": "08:00",
        "period_end": "20:00"
      },
      {
        "weekday": 2,
        "period_begin": "08:00",
        "period_end": "20:00"
      },
      {
        "weekday": 3,
        "period_begin": "08:00",
        "period_end": "20:00"
      },
      {
        "weekday": 4,
        "period_begin": "08:00",
        "period_end": "20:00"
```

```
      },
      {
        "weekday": 5,
        "period_begin": "08:00",
        "period_end": "20:00"
      }
    ],
    "twentyfourseven": false,
    "exceptional_openings": [
      {
        "period_begin": "2014-06-21T09:00:00Z",
        "period_end": "2014-06-21T12:00:00Z"
      }
    ],
    "exceptional_closings": [
      {
        "period_begin": "2014-06-24T00:00:00Z",
        "period_end": "2014-06-25T00:00:00Z"
      }
    ]
  }
```

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

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	-	-

5.4.37. ReservationStatus *enum*

Error codes for reservation related errors.

Value	**Description**
CANCELED_RESERVATION	The Reservation has been canceled by the CPO.
DEVICE_OFFLINE	Reservation for the requested EVSE requires remote communication to a charging device and this remote communication link is not available.
EVSE_OCCUPIED	EVSE is currently occupied, another session is ongoing. Cannot start a new session.
EVSE_INOPERATIVE	EVSE is currently inoperative or faulted.
NOT_SUPPORTED	Reservations are not supported by the receiving Party or on the Location for which one was requested.
UNKNOWN_RESERVATION	The Reservation in the requested command is not known by the receiving Party.

5.4.38. ReserveNowRequest *class*

Property	Type	Card.	Description
token	Token	1	The Token for which Party Y should reserve an EVSE
expiry_date	DateTime	1	The point in time at which this reservation ends
reservation_id	CiAsciiString[1..36]	1	An identifier for this reservation
location_id	CiAsciiString[1..36]	1	Location ID of the Location (belonging to the CPO this request is sent to) for which to reserve an EVSE.
evse_uid	CiAsciiString[1..36]	?	Optional EVSE UID of the EVSE to reserve if a specific EVSE has to be reserved.
authorization_reference	CiAsciiString[1..36]	?	Reference to the authorization given by the eMSP. When given, this reference will be provided in the Session and/or CDR that may eventually be transferred from Party Y to Party X as a result of this reservation request.

5.4.39. ReservationError *class*

Property	Type	Card.	Description
status	ReservationStatus	1	An error code that signals why the reservation request failed
message	DisplayText	*	Human-readable description of the reason for the status (if one can be provided), multiple languages can be provided.

5.4.40. ResetEvseRequest *class*

Property	Type	Card.	Description
location_id	CiAsciiString[1..36]	1	The ID of the Location at which a device is to be reset
evse_uid	CiAsciiString[1..36]	1	The value of the uid field of the EVSE of this Location for which the device is requested to be reset

5.4.41. LocationService *enum*

An enumeration of services that are offered at the Location by the CPO or affiliated parties.

Value	Description
ACCESSIBLE_CHARGING	One or more EVSEs have accessibility modifications in place to allow use by people with disabilities. Note that more information on accessibility modifications can be provided using the various fields for images and in the `parking` field of the EVSE object.
ASSISTANCE	Assistance from on-site staff is available to help a Driver charge at the Location.
CAMERA_SURVEILLANCE	Security monitoring with video cameras is in place at the Location.

Value	Description
EMERGENCY_CALL	A voice communication channel is available for the Driver to contact security staff from the Location.
WIFI	WLAN Internet connectivity is available at the Location.

5.4.42. Status *enum*

The status of an EVSE.

Value	Description
AVAILABLE	The EVSE/Connector is able to start a new charging session.
BLOCKED	The EVSE/Connector is not accessible because of a physical barrier, i.e. a car.
CHARGING	The EVSE/Connector is in use.
INOPERATIVE	The EVSE/Connector is not yet active, or temporarily not available for use, but not broken or defect.
OUTOFORDER	The EVSE/Connector is currently out of order, some part/components may be broken/defect.
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).

5.4.43. StatusSchedule *class*

This type is used to schedule status periods in the future. The eMSP can provide this information to the EV user for trip planning purposes. A period MAY have no end. A period without end can be used for example that after construction completes tomorrow, a charging station will be available indefinitely.

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

NOTE The scheduled status is purely informational. When the status actually changes, the CPO must push an update using the EVSE Status module just like for status changes that were not scheduled.

5.4.44. UnlockConnectorRequest *class*

Property	Type	Card.	Description
location_id	CiAsciiString[1..36]	1	The ID of the Location at which a Connector is to be unlocked

Property	Type	Card.	Description
evse_uid	CiAsciiString[1..36]	1	The value of the uid field of the EVSE of this Location of which it is requested to unlock the Connector.
connector_id	CiAsciiString[1..36]	1	Identifier of the Connector of this Location of which it is requested to unlock. This is the identifier found in the `id` field of the Connector object.

5.4.45. VehicleType *enum*

A categorization of vehicles to indicate which type of vehicles can use a certain EVSE.

Value	Description
MOTORCYCLE	A motorcycle
PERSONAL_VEHICLE	A personal vehicle, a passenger car
PERSONAL_VEHICLE_WITH_TRAILER	A personal vehicle with a trailer attached
VAN	A light-duty van with a height smaller than 275 cm
TRUCK	A heavy-duty truck without a trailer
TRUCK_WITH_TRAILER	A heavy-duty truck with a trailer attached
BUS	A bus or a motor coach
DISABLED	A vehicle with a permit for parking spaces for people with disabilities

NOTE

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

6. EVSE Status

This chapter describes the EVSE Status module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the EVSE Status module, the type of data object is the EVSE Status, a small object that describes the current availability status of a particular EVSE. That is, these objects answer the question: can a Driver currently charge come to this EVSE to charge there, and if not, why not?

This module is meant to be used together with the Locations module. The EVSE Status objects exchanged with the EVSE Status module reference EVSEs exchanged with the Locations module. A Party that wants to know about all the Locations of another Party and the statuses of the EVSEs of these Locations, will subscribe to both the Locations and EVSE Status modules of that other Party.

Historically, the EVSE Status module was split off from the Locations module during the development of OCPI 3.0 based on OCPI 2.2.1. In OCPI 2.2.1 and earlier OCPI versions, the status of EVSEs was part of the Location objects.

The EVSE Status module was split off from the Locations module in order to allow Parties who are not interested in live availability status information to not receive status updates. EVSE Status updates are very frequent and thus processing all changes in EVSE Status in the network of a large CPO comes with costs and challenges that some Locations subscribers may not be willing to bear.

6.1. Replicating EVSE Status objects

6.1.1. UC: 06.01 - Replicate EVSE status objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains and maintains up-to-date information of the availability of the charging infrastructure offered by Party Y
2	**Description**	Using the use cases of the Party-Issued Objects chapter, EVSE status objects are replicated from Party Y to Party X
3	**Actors**	eMSP, CPO, NAP, NSP
4	**Flow**	1. Party X subscribes to Party Y's EVSE status objects 2. Party Y pushes all their EVSE status objects as of subscription time to Party X 3. Party Y pushes every newly updated EVSE status object to Party X as soon as these updates happen 4. This continues until either party cancels the subscription
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's EVSE Status module to Party X on Platform A.
6	**Postconditions**	Party X has up-to-date information on the EVSE Status objects of Party Y
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 32. UC: 06.01 Requirements

ID	Precondition	Requirement
R.06.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "evsestatuses" as the ModuleID value.
R.06.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.06.01.01
R.06.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.06.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the EvseStatus object type.
R.06.01.04	Platform B sends a Party Issued Update update to Party Y on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.06.01.01	The identifier of the Party Issued Object, that is the values of the `id` field of the PartyIssuedObjectUpdate objects, SHALL be set to the EVSE UIDs of the EVSE that the status updates apply to. This is how EVSE Statuses are related to the EVSEs from the Locations module.

NOTE Parties subscribing to both the Locations and the EVSE Statuses of another Party may receive Locations with EVSEs for which they have not yet received a status. They may also receive EVSE Statuses for EVSEs that they have not yet received as part of a Location from the Locations module. This is a logical consequence of splitting the Locations and EVSE Status modules and it is something that subscribers will have to deal with on their side.

6.2. Remote Procedure Calls on EVSE Status objects

This module does not specify any Remote Procedure Calls.

6.3. Object type definitions

6.3.1. EvseStatus *class*

Property	Type	Card.	Description
status	Status	1	The current status of the EVSE
timestamp	DateTime	?	The timestamp of the moment that this Status came into effect, as reported by the device. The sender platform SHOULD NOT include this field unless a timestamp for the status change was reported by the remote device containing the EVSE.

7. Sessions

This chapter describes the Sessions module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the Sessions module, the type of data object is the Session. The name Session is short for Charging Session. What a Charging Session is is defined in the Terminology section of the Business Use Cases document. How a Charge Session is represented in OCPI messages is specified below.

When a CPO registers a charging Session in their systems, they push a Session object to the eMSP that issued the Token that the Session was started with.

Sessions cannot be deleted. Instead Sessions become finalized and immutable when their status becomes `COMPLETED`.

For a lot of smart charging use cases, input from the driver is needed. The smart charging algorithms need to be able to give certain Session priority over others. In other words they need to know how much energy an EV needs before what time. The CPO can learn about the driver's smart charging preferences via the `default_profile_type` field in the Token object for the token used to start the Session. Alternatively the CPO can tell the MSP about the Driver's preferences while the Session is ongoing with the Change Charging Preferences functional use case .

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

Via Tariffs and Tariff Associations the CPO can give different Charging Preferences different prices.

When an EV driver makes a Reservation for a Charging Station/EVSE, the Sender SHALL create a new Session object with `status` = `RESERVED`.

When a reservation results in a charging Session for the same Token, the Session's status changes to `ACTIVE`.

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

A CDR might be created even if no energy was transferred to the EV and no EV was ever present at or connected to the Charging Station, just for the costs of the reservation.

7.1. Changes from OCPI 2.2.1

- Removed the country code and party ID, as from all Party Issued Objects, because these are now unambiguously transferred using the Party Issued Object replication mechanism.
- Moved the StartSession and StopSession commands from the Commands module to the Sessions module.
- Included a "displayTariff" field in the StartSession command to allow an eMSP to tell the CPO what the eMSP will charge the Driver.
- Added functionalities to let the eMSP make the CPO display a message on the Charging Station and to let the CPO make the eMSP send a message to the Driver.
- Renamed `kwh` field to `energy`.
- Added `tariff_id` and `tariff_association_id` fields to Session object.

7.2. Replicating Session objects

7.2.1. UC: 07.01 - Replicate Session objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains and maintains an up-to-date copy of the Session objects that are issued by Party Y on a Platform B
2	**Description**	Using the use cases of the Party-Issued Objects chapter, Session objects are replicated from Party Y to Party X
3	**Actors**	eMSP, CPO, Hub
4	**Flow**	1. Party X subscribes to Party Y's Session objects 2. Party Y pushes every newly updated Session object to Party X as soon as these updates happen 3. This continues until either party cancels the subscription
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Sessions to Party X on Platform A.
6	**Postconditions**	Party X has up-to-date information on the Charge Sessions issued by Party Y
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 33. UC: 07.01 Requirements

ID	Precondition	Requirement
R.07.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "sessions" as the ModuleID value.
R.07.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.07.01.01
R.07.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.07.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the Session object type.

7.3. Remote Procedure Calls on Session Objects

7.3.1. UC: 07.02 - Start a Session

1	**Objective(s)**	**1.** Party X initiates a Charging Session on an EVSE at a Location issued by Party Y.
2	**Description**	This use case is meant for situations where another company requests a CPO to start a Session. The most common example is probably a Driver using a mobile phone app provided by their eMSP to initiate a Charge Session. Another example would be a third party providing customer support to the CPO, using a back-office UI to a start a Session for a Driver who called them.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains parameters indicating on which EVSE a Session is to be started and which Charge Token should be associated with the Session. 2. Party Y determines if it is willing to indeed start a Charging Session as requested. 3. Platform B responds to Platform A on behalf of Party Y to inform Party X if their request was accepted. 4. If the request was accepted, Party Y proceeds to start the Charging Session. 5. If the request was accepted, Platform B sends a request to Platform A on behalf of Party Y to inform Party X of the result of the session start request to the Charging Station.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Locations to Party X on Platform A. Platform B serves Party Y's Sessions to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to start a Session at Party Y's Locations. * Party Y attempted to start a Session on a Charging Station at one of Party Y's Locations at Party X's request. Also Party X knows whether this attempt was successful or not.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	Party X will typically track the outcome of an attempt to start a Session by monitoring for the three steps of execution that are communicated via OCPI: 1. A synchronous response is received, indicating that the CPO received and understood the request. 2. An asynchronous response is received, indicating whether the charging device will actually attempt to execute the request. 3. A Session object is received via Party Issued Object replication, indicating that the charging device was successful in its attempt and the Session has begun.

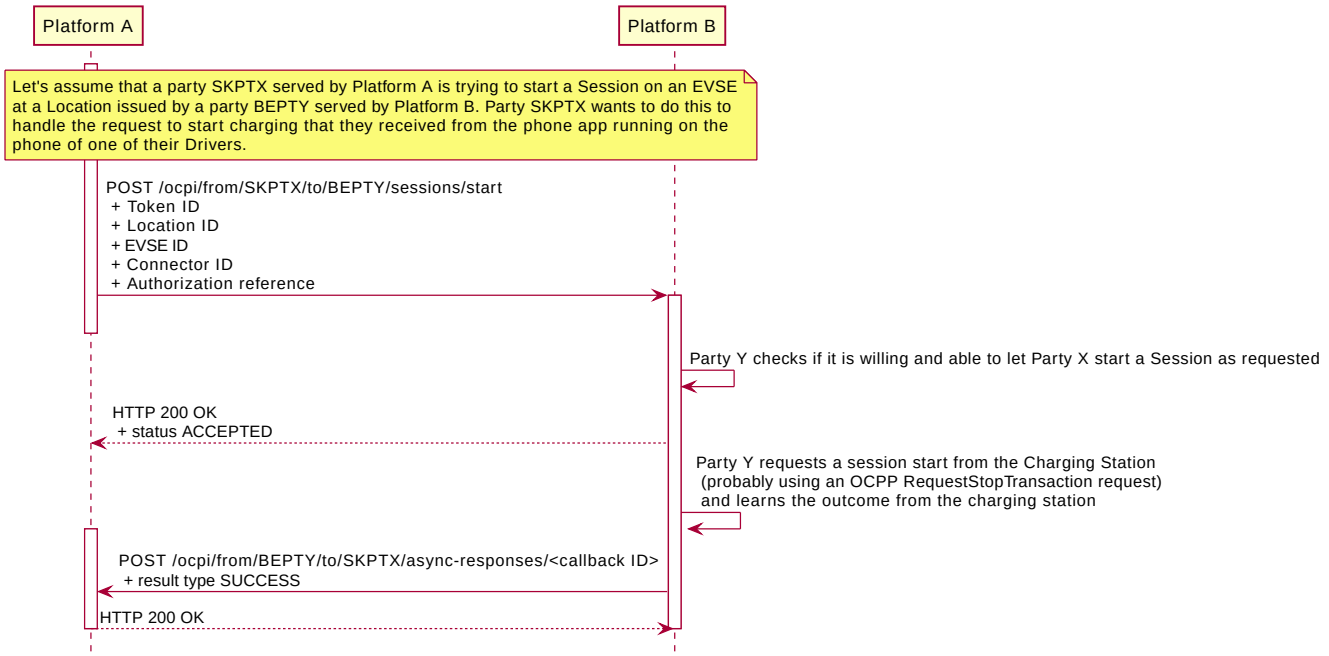


Figure 39. Sequence Diagram: Start a Session

Table 34. UC: 07.02 Requirements

ID	Precondition	Requirement
R.07.02.01		Platform A SHALL make the request to start the Session on an EVSE following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.02.02		Platform A SHALL use "start" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.02.03		Platform A SHALL use a StartSessionRequest in the payload field of the in the payload field of AsyncRequest object in the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.02.04	Party Y cannot or will not act upon the start request because of the status of the EVSE or the Connector or the device containing the EVSE and the Connector	Platform B SHALL respond to Platform A's request with a response with ACCEPTED in the data field of the OcpiResponse object. Platform B SHALL then send a request with an AsyncResponse object with the payload field unset and with the error field set to an appropriate SessionCommandError object.
R.07.02.05	Party Y received a confirmation from the Charging Station that the start request will be executed	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse object set to SUCCESS, and the payload and error fields of this AsyncResponse left unset.
R.07.02.06	The `evse_uid` field of the StartSessionRequest is not filled in in the request from Platform A	Party Y or the Charging Station can itself decide on which EVSE to start a new session. This may not be supported on all Charging Stations.
R.07.02.07	A Session is started by Party Y as a result of the request from Platform A	The Session SHALL be issued by Party Y with the token set to the Token provided by Party X.

ID	Precondition	Requirement
R.07.02.08		Party Y SHALL NOT check the validity of the Token provided before sending the request to the Charging Station. That is, Party X's using the Token in a StartSessionReequest already informs Party Y that the Token is valid according to Party X at the moment Party X starts this use case.
R.07.02.09	Party Y receives a request from the Charging Station to check the validity of the Token, e.g. an OCPP Authorize request.	- If this Token is of type: AD_HOC_USER or APP_USER the CPO SHALL NOT do a realtime authorization with Party X for this Token. - If this Token is of type: RFID, the CPO SHALL NOT do a realtime authorization with Party X for this Token at the given EVSE/Charging Station within 15 minutes after having received this StartSession. This means that if the driver decided to use his RFID within 15 minutes at the same Charging Station, because the app is not working somehow, the RFID is already authorized.
R.07.02.10		Party X MAY use a Token that has not been pushed via the Token module. This typically happens with AD_HOC_USER or APP_USER Tokens, which are only used in start and stop requests sent by an eMSP. These are never used locally at the Charging Station like RFID.
R.07.02.11	Party X sent a Token in the StartSessionRequest that they did not issue to Party Y using Party Issued Object replication.	Platform B SHALL NOT store this token in their database of Tokens issued by Party X to Party Y.
R.07.02.12	Party X sent a Token in the StartSessionRequest that they did not issue to Party Y using Party Issued Object replication.	The information of the Token SHALL be included in the Session and CDR objects about the session started for Party X, despite the Token's not being stored in the Party Issued Object replication dataset as per R.07.02.11.
R.07.02.13		Party X SHALL NOT use Tokens for this use case that were issued to Party X by another Party. That is, Party X shall not start a session on one of Party Y's Locations with a Token from a third Party.

7.3.2. UC: 07.03 - Stop a Session

1	**Objective(s)**	**1.** Party X stops a Charging Session on an EVSE at a Location issued by Party Y.
2	**Description**	This use case is meant for situations where another company requests a CPO to stop a Session. The most common example is probably a Driver using a mobile phone app provided by their eMSP to stop a Charge Session. Another example would be a third party providing customer support to the CPO, using a back-office UI to a stop a Session for a Driver who called them.
3	**Actors**	eMSP, CPO

4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the identifier of the session that is to be stopped. 2. Party Y determines if it is willing to indeed stop the Charging Session. 3. Platform B responds to Platform A on behalf of Party Y to inform Party X if their request was accepted. 4. If the request was accepted, Party Y proceeds to stop the Charging Session. 5. If the request was accepted, Platform B sends a request to Platform A on behalf of Party Y to inform Party X of the result of the session stop request to the Charging Station.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Sessions to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to stop a Session at Party Y's Locations. * Party Y attempted to stop a Session on a Charging Station at one of Party Y's Locations at Party X's request. Also Party X knows whether this attempt was successful or not.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	Party X will typically track the outcome of an attempt to stop a Session by monitoring for the three steps of execution that are communicated via OCPI: 1. A synchronous response is received, indicating that the CPO received and understood the request. 2. An asynchronous response is received, indicating whether the charging device will actually attempt to execute the request. 3. A Session object update is received via Party Issued Object replication, indicating that the charging device was successful in its attempt and the Session status has changed to `COMPLETED`.

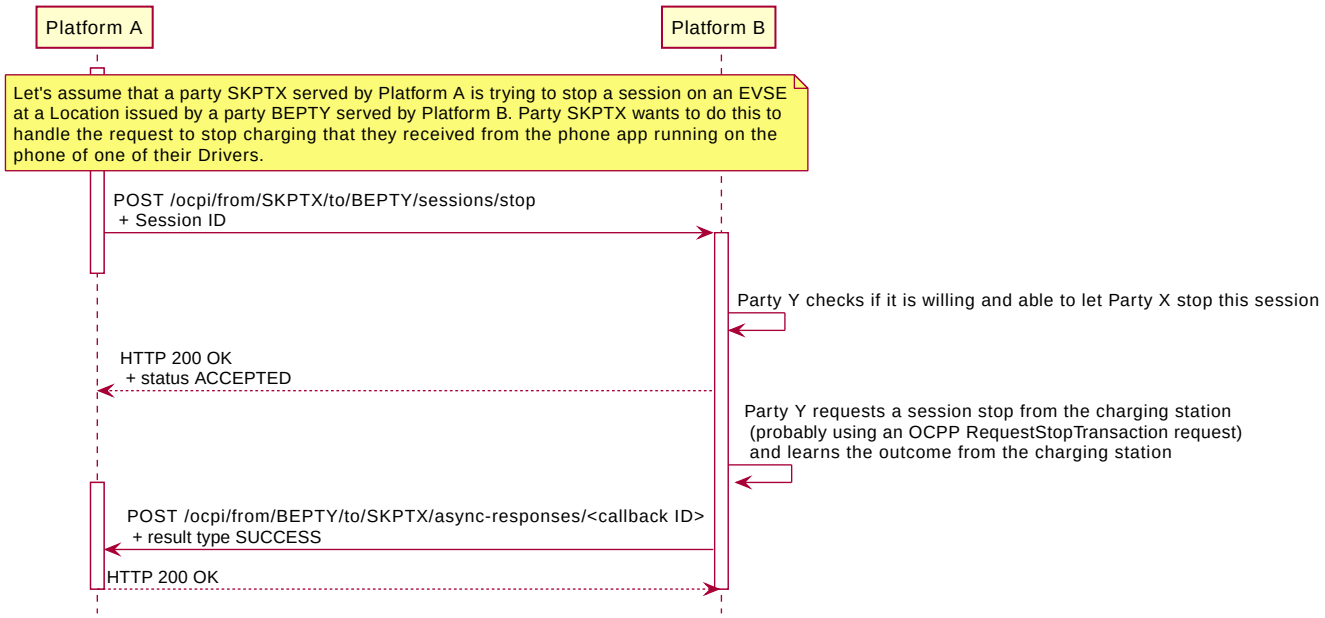


Figure 40. Sequence Diagram: Stop a Session

Table 35. UC: 07.03 Requirements

ID	Precondition	Requirement
R.07.03.01		Platform A SHALL make the request to stop the Session on an EVSE following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.03.02		Platform A SHALL use "stop" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.03.03		Platform A SHALL use a StopSessionRequest in the payload field of the in the payload field of AsyncRequest object in the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.03.04	Party Y cannot or will not act upon the stop request because of the status of the EVSE or the Connector or the device containing the EVSE and the Connector	Platform B SHALL respond to Platform A's request with a response with ACCEPTED in the data field of the OcpiResponse object. Platform B SHALL then send a request with an AsyncResponse object with the payload field unset and with the error field set to an appropriate SessionCommandError object.
R.07.03.05	Party Y got a confirmation from the Charging Station that the stop request will be executed	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse object set to SUCCESS, and the payload and error fields of this AsyncResponse left unset.

7.3.3. UC: 07.04 - Change Charging Preferences

1	**Objective(s)**	**1.** Party Y switches to a different objective for smart charging optimization for a Session in which a Driver related to Party X is charging.
2	**Description**	This use case allows the eMSP (Party X) to inform the CPO (Party Y) of what the eMSP would like the CPO to optimize for with smart charging. This is done by sending a ProfileType value to the CPO.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the ProfileType that Party X would like Party Y to use for the session. 2. Party Y determines if it is willing to switch to the requested profile type. 3. If Party Y switches if it is indeed willing to. 4. Platform B responds to Platform A on behalf of Party Y to inform Party X if their request was executed.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Sessions to Party X on Platform A.

6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to change the charging preferences. * Party Y changed the charging preferences that are optimized for in the smart charging of the Session.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

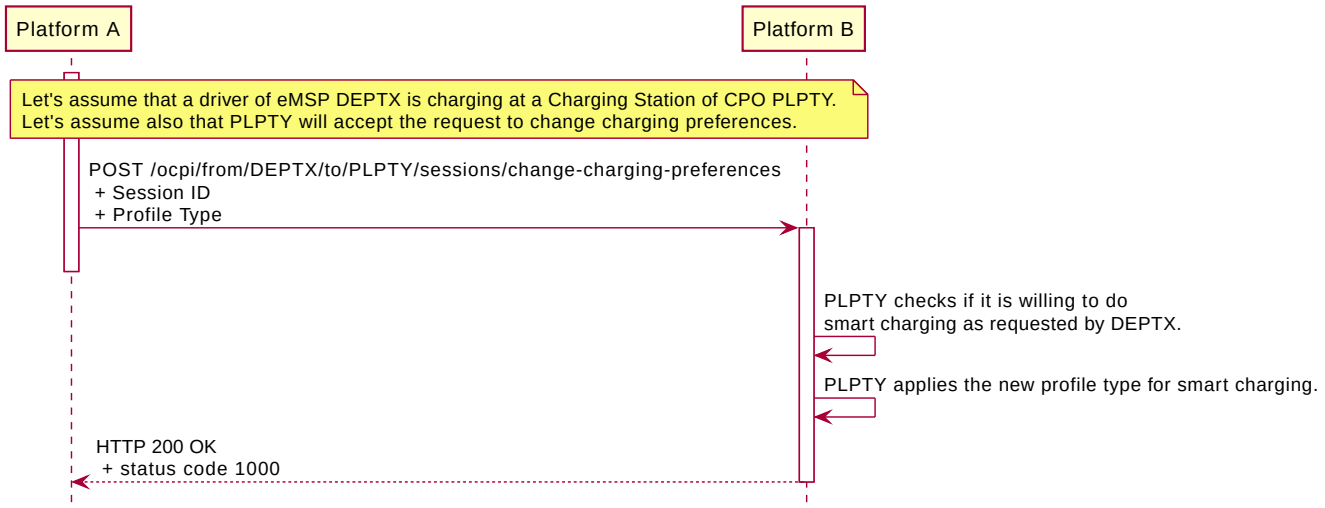


Figure 41. Sequence Diagram: Change Charging Preferences

Table 36. UC: 07.04 Requirements

ID	Precondition	Requirement
R.07.04.01		Platform A SHALL make the request to reserve an EVSE following Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.04.02		Platform A SHALL use "change-charging-preferences" as the operation name for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.04.03		Platform A SHALL use POST as the HTTP request verb when making its request.
R.07.04.04		Platform A SHALL use a ChangeChargingPreferencesRequest in the request body for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.04.05		Platform B SHALL include a ChangeChargingPreferencesResponse in the `data` field of the OcpiResponse object in the response body according to Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.

7.3.4. UC: 07.05 - Notify Session receiver of the active Charging Profile

1	**Objective(s)**	**1.** Party X (typically an eMSP) learns the active Charging Profile that Party Y (typically a CPO) is using for a Charging Session, so that Party X can inform the Driver of the charging speed that they should expect.
2	**Description**	The CPO (Party Y) makes a remote procedure call to the eMSP (Party X) to send them the active Charging Profile.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform B makes a request on behalf of Party Y to Platform A receiving the request on behalf of Party X. This request contains the ChargingProfile that Party Y is using for the Charging Session. 2. Platform A sends a response to Platform B acknowledging that it received the request.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Sessions to Party X on Platform A.
6	**Postconditions**	Party X knows the active Charging Profile that Party Y is using for a certain Charging Session.
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

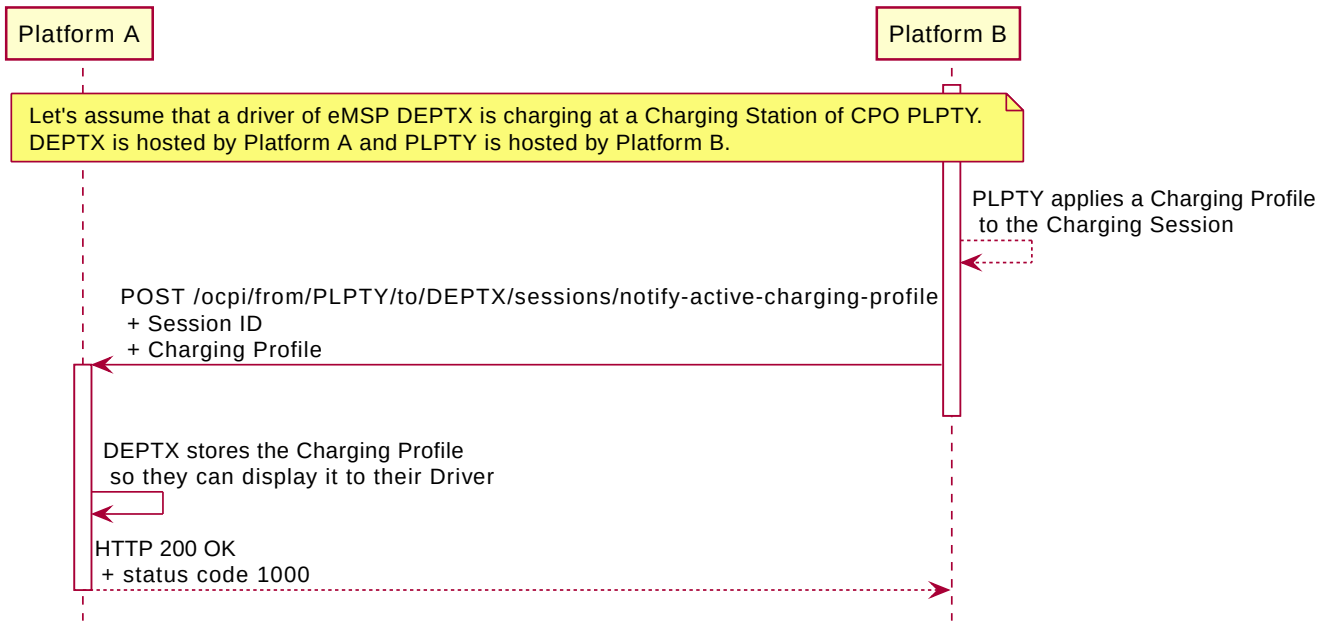


Figure 42. Sequence Diagram: Notify Session receiver of the active Charging Profile

Table 37. UC: 07.05 Requirements

ID	Precondition	Requirement
R.07.05.01		Platform B SHALL make the request to notify Platform A of the current Charging Profile following Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.05.02		Platform B SHALL use "notify-active-charging-profile" as the operation name for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.05.03		Platform B SHALL use POST as the HTTP request verb when making its request.
R.07.05.04		Platform B SHALL use a NotifyActiveChargingProfileRequest in the request body for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.05.05		Platform A SHALL leave the `data` field unset in the OcpiResponse object in the response body according to Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.

7.3.5. UC: 07.06 - Send Message for Driver About Session to eMSP

1	**Objective(s)**	**1.** An eMSP receives a message for a Driver of theirs from a CPO so that they can notify the Driver of some event at the Charging Station.
2	**Description**	The CPO (Party Y) makes a remote procedure call to the eMSP (Party X) to send them the message for the Driver.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform B makes a request on behalf of Party Y to Platform A receiving the request on behalf of Party X. This request contains the message that the CPO wants the Driver to see. 2. Platform A sends a response to Platform B acknowledging that it received the request.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Sessions to Party X on Platform A. The session about which Party Y wants to send a message has been replicated from Party Y to Party X.
6	**Postconditions**	Party X has a message from Party Y. Party X knows that Party Y wants the Driver to see the message.
7	**Error handling**	Error reporting by Platform A follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

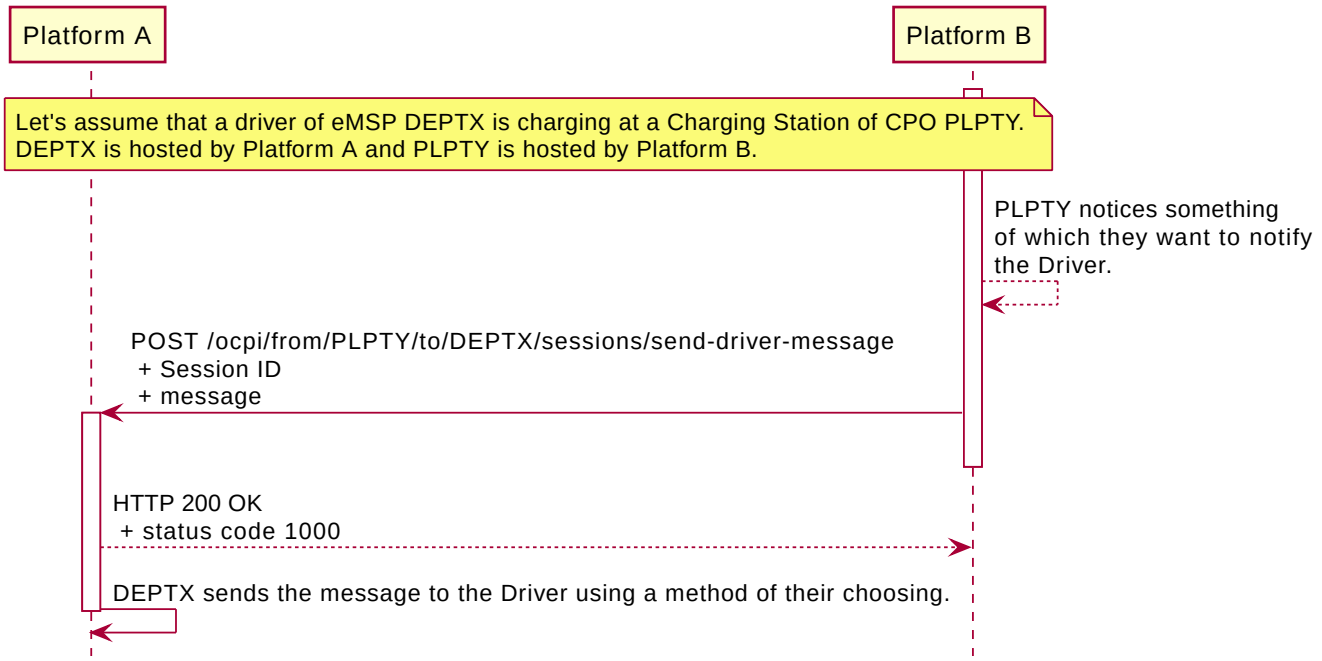


Figure 43. Sequence Diagram: Send Message for Driver About Session to eMSP

Table 38. UC: 07.06 Requirements

ID	Precondition	Requirement
R.07.06.01		Platform B SHALL make the request to send Platform A a message for a Driver according to Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.06.02		Platform B SHALL use "send-driver-message" as the operation name for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.06.03		Platform B SHALL use POST as the HTTP request verb when making its request.
R.07.06.04		Platform B SHALL use a NotifyActiveChargingProfileRequest in the request body for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.06.05		Platform A SHALL leave the `data` field unset in the OcpiResponse object in the response body according to Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.07.06.06	The Session ID in the SendDriverMessageRequest in the request body identifies a Session by a Driver of Party X	Party X SHOULD try to notify the Driver by communication methods that they have at their disposal, like for example a mobile push notification or an email.
R.07.06.07		Party X SHOULD NOT wait for the delivery of the message to the Driver as in R.07.06.06 before responding to Party Y's request.

7.3.6. UC: 07.07 - Send Message for Driver About Session to CPO

1	**Objective(s)**	**1.** An eMSP sends a message for a Driver of theirs to a CPO so that the CPO can display the message on the Charging Station.
2	**Description**	The eMSP (Party X) makes a remote procedure call to the CPO (Party Y) to send them the message to be displayed on the Charging Station.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the message that the eMSP wants the CPO to display on the Charging Station. 2. Platform B sends a response to Platform A acknowledging that it received the request.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Sessions to Party X on Platform A. The session about which Party X wants to send a message has been replicated from Party Y to Party X.
6	**Postconditions**	Party Y has a displayed from Party X on the Charging Station where a Driver of Party X's is charging. or Party X knows that Party Y cannot or will not display the message on their Charging Station.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

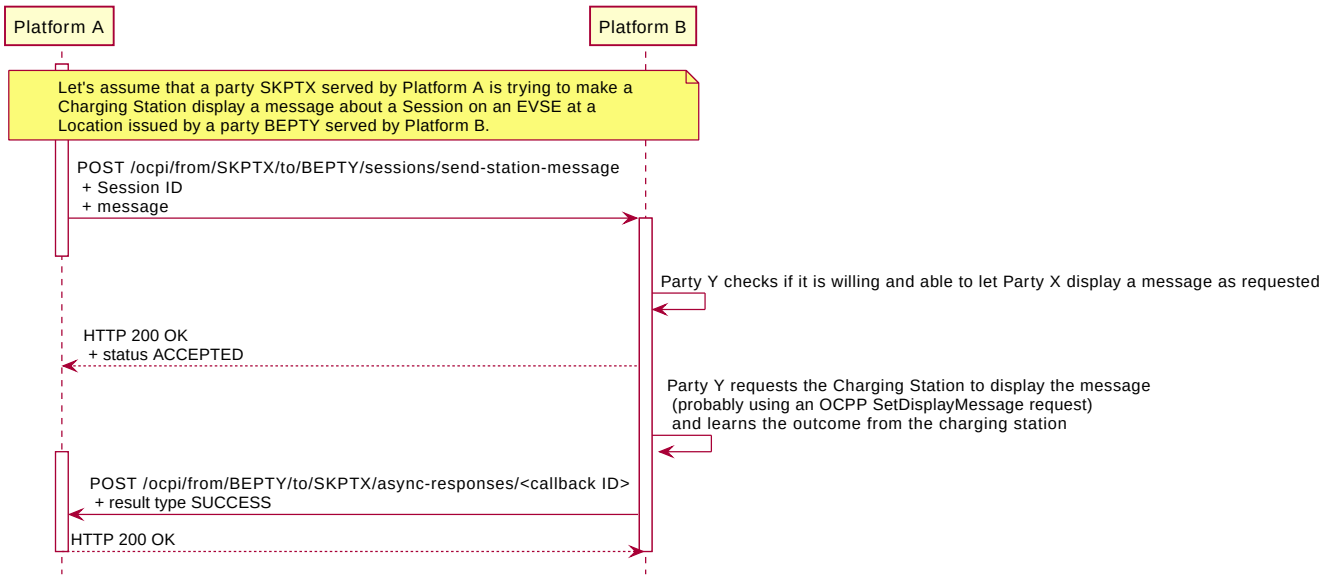


Figure 44. Sequence Diagram: Send Message for Driver About Session to CPO

Table 39. UC: 07.07 Requirements

ID	Precondition	Requirement
R.07.07.01		Platform A SHALL make the request to display the message on a Charging Station following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.07.02		Platform A SHALL use "send-station-message" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.07.03		Platform A SHALL use a SendDriverMessageRequest in the payload field of the in the payload field of AsyncRequest object in the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.07.07.04	Party Y got a confirmation from the Charging Station that the request to display a message will be executed	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse object set to SUCCESS, and the payload and error fields of this AsyncResponse left unset.

7.4. Data types

7.4.1. ChargingPreferences *class*

Contains the charging preferences of an EV driver.

Property	Type	Card.	Description
session_id	CiAsciiString[1..36]	1	The identifier of the session for which charging preferences are to be set.
profile_type	ProfileType	1	Type of Smart Charging Profile selected by the driver. The ProfileType has to be supported at the Connector and for every supported ProfileType, a Tariff MUST be provided. This gives the EV driver the option between different pricing options.
departure_time	DateTime	?	Expected departure. The driver has given this Date/Time as expected departure moment. It is only an estimation and not necessarily the Date/Time of the actual departure.
energy_need	number	?	Requested amount of energy in kWh. The EV driver wants to have this amount of energy charged.
discharge_allowed	boolean	?	The driver allows their EV to be discharged when needed, as long as the other preferences are met: EV is charged with the preferred energy (`energy_need`) until the preferred departure moment (`departure_time`). Default if omitted: **false**

7.4.2. ChargingPreferencesResponse *enum*

An enum with possible responses to a Change Charging Preferences RPC call.

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

7.4.3. NotifyActiveChargingProfileRequest *class*

A request made by a Party who is a Session sender to notify the Session receiver of a Charging Profile that is being applied to a Session.

Property	Type	Card.	Description
session_id	CiAsciiString[1..36]	1	The ID of the Session to which the Charging Profile applies.
charging_profile	ChargingProfile	1	The Charging Profile that applies to the Session.

7.4.4. SendDriverMessageRequest *class*

A request made by one Party who is a Session sender or receiver to notify another Party of something that the one Party wants the other Party to show to the Driver.

Property	Type	Card.	Description
session_id	CiAsciiString[1..36]	1	The ID of the Session for which a message is to be shown to the Driver.
messages	DisplayText	+	Messages to be shown. There can be multiple DisplayText values so that the Session sender can send the same message in different languages.

7.4.5. Session *class*

The Session object describes one charging session. That doesn't mean it is required that energy has been transferred between EV and the Charging Station. It is possible that the EV never took energy from the Charging Station because

it was instructed not to take energy by the driver. But as the EV was connected to the Charging Station, some form of start tariff, park tariff or reservation cost might be relevant.

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

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

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

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

Property	Type	Card.	Description
start_date_time	DateTime	1	The timestamp when the session became ACTIVE in the Charging Station. When the session is still PENDING, this field SHALL be set to the time the Session was created at the Chrging Station. When a Session goes from PENDING to ACTIVE, this field SHALL be updated to the moment the Session went to ACTIVE in the Charging Station.
end_date_time	DateTime	?	The timestamp when the session was completed/finished, charging might have finished before the session ends, for example: EV is full, but parking cost also has to be paid.
energy	decimal	1	How many kWh of energy were transferred through the EVSE into the vehicle.
cdr_token	CdrToken	1	Token used to start this charging session, including all the relevant information to identify the unique token.
auth_method	AuthMethod	1	Method used for authorization. This might change during a session. This can happen for example when the session was started with a reservation according to use case Reserve an EVSE at a Location. Initially the authorization method will be `COMMAND` which changes to `WHITELIST` when the driver arrives and starts charging using a Token that is whitelisted.

Property	Type	Card.	Description
authorization_reference	CiAsciiString[1..36]	?	Reference to the authorization given by the eMSP. When the eMSP provided an `authorization_reference` in either: real-time authorization, Start a Session or Reserve an EVSE at a Location this field SHALL contain the same value. When different `authorization_reference` values have been given by the eMSP that are relevant to this Session, the last given value SHALL be used here.
location_id	CiAsciiString[1..36]	1	Location ID of the Location object of this CPO, on which the charging session is/was happening.
connector	SessionConnector	?	The Connector that the Session happened on. This is allowed to be unset if and only if the Session is created for a reservation for which no EVSE has been assigned yet.
meter_id	AsciiString[1..255]	?	Optional identification of the kWh meter.
currency	CiAsciiString[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.
tariff_association_id	Tariff	1	The ID of the Tariff Association that was used to look up the Tariff of this Session. When the session is free, the ID of a Tariff Association for a *Free of Charge* tariff is to be given in this field.
tariff_id	Tariff	1	The ID of the Tariff that was used to compute what this Session costs. When the session is free, the ID of a *Free of Charge* tariff is to be given in this field.
total_cost	Price	?	The total cost of the session in the specified currency. This is the price that the eMSP will have to pay to the CPO. A total_cost of 0.00 means free of charge. When omitted, i.e. no price information is given in the Session object, it does not imply the session is/was free of charge.
status	SessionStatus	1	The status of the session.
last_updated	DateTime	1	Timestamp when this Session was last updated (or created).

7.4.5.1. Examples

Simple Session example of just starting a session

```
{
    "start_date_time": "2020-03-09T10:17:09Z",
    "energy": 0.0,
    "cdr_token": {
        "country_code": "NL",
        "party_id": "TST",
        "uid": "123abc",
        "type": "RFID",
        "contract_id": "NL-TST-C12345678-S"
```

```
    },
    "auth_method": "WHITELIST",
    "location_id": "LOC1",
    "evse_uid": "3256",
    "connector_id": "1",
    "tariff_association_id": "TAC1",
    "tariff_id": "TAR1",
    "currency": "EUR",
    "total_cost": {
        "before_taxes": 2.5
    },
    "status": "PENDING",
    "last_updated": "2020-03-09T10:17:09Z"
}
```

Simple Session example of a short finished session

```
{
  "start_date_time": "2015-06-29T22:39:09Z",
  "end_date_time": "2015-06-29T23:50:16Z",
  "energy": 41.12,
  "cdr_token": {
        "country_code": "NL",
        "party_id": "TST",
        "uid": "123abc",
        "type": "RFID",
        "contract_id": "NL-TST-C12345678-S"
  },
  "auth_method": "WHITELIST",
  "location_id": "LOC1",
  "evse_uid": "3256",
  "connector_id": "1",
  "tariff_association_id": "TAC1",
  "tariff_id": "TAR1",
  "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": "TIME",
      "volume": 0.718
    }]
  }],
  "total_cost": {
    "before_taxes": 8.50,
    "taxes": [{
      "name": "VAT",
      "amount": 0.85
    }]
  },
```

```
  "status": "COMPLETED",
  "last_updated": "2015-06-29T23:50:17Z"
}
```

7.4.6. ProfileType *enum*

Different smart charging profile types.

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

7.4.7. SessionCommandError *class*

Property	Type	Card.	Description
status	SessionCommandStatus	1	An error code that signals why the requested operation was not executed or failed
message	DisplayText	*	Human-readable description of the reason for the status (if one can be provided), multiple languages can be provided.

7.4.8. SessionCommandStatus *enum*

Value	Description
DEVICE_OFFLINE	The operation could not be performed because remote communication is not available to the device that has to execute the operation.
EVSE_OCCUPIED	The operation could not be performed because the EVSE that the operation was requested on is occupied.
EVSE_INOPERATIVE	The operation could not be performed because the EVSE that the operation was requested on is inoperative.

7.4.9. SessionConnector *class*

A reference to the Connector that a Session happened on.

Property	Type	Card.	Description
evse_uid	CiAsciiString[1..36]	1	EVSE.uid of the EVSE of this Location on which the charging session is/was happening.
connector_id	CiAsciiString[1..36]	1	Connector ID of the Connector of this Location where the charging session is/was happening.

7.4.10. SessionStatus *enum*

Defines the state of a session.

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

7.4.11. StartSession *class*

Property	Type	Card.	Description
response_url	URL	1	URL that the CommandResult POST should be sent to. This URL might contain a unique ID to be able to distinguish between StartSession requests.
token	Token	1	Token object the Charging Station has to use to start a new session. The Token provided in this request is authorized by the eMSP.
location_id	CiAsciiString[1..36]	1	Location ID of the Location (belonging to the CPO this request is sent to) on which a session is to be started.
evse_uid	CiAsciiString[1..36]	?	Optional EVSE uid of the EVSE of this Location on which a session is to be started. Required when `connector_id` is set.
connector_id	CiAsciiString[1..36]	?	Optional Connector ID of the Connector of the EVSE on which a session is to be started. This field is required when the capability: START_SESSION_CONNECTOR_REQUIRED is set on the Charging Station.
authorization_reference	CiAsciiString[1..36]	?	Reference to the authorization given by the eMSP, when given, this reference will be provided in the relevant Session and/or CDR.

Property	Type	Card.	Description
display_tariff	Tariff	?	The Tariff that will be charged by the eMSP to the Driver, to be displayed on the Charging Station. This added because regulations in the US State of California require that a Driver see on the Charging Station what they will be paying when they start a Charging Session. This field can also be used in combination with IEC 15118 to provide pricing information from the Charging Station to the vehicle. Where these two use cases do not apply, this field may be left empty.

NOTE In case of an OCPP 1.x Charging Station, the EVSE ID should be mapped to the connector ID of a Charging Station. OCPP 1.x does not have good support for Charging Stations that have multiple connectors per EVSE. To make StartSession over OCPI work, the CPO SHOULD present the different connectors of an EVSE as separate EVSE, as is also written by the OCA in the application note: "Multiple Connectors per EVSE in a OCPP 1.x implementation".

7.4.12. StopSessionRequest *class*

Property	Type	Card.	Description
session_id	CiAsciiString[1..36]	1	Session ID of the Session that is requested to be stopped.

8. CDRs

8.1. Introduction

This chapter describes the CDRs module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the CDRs module, the type of data object is the CDR. CDR stands for Charge Detail Record.

A **Charge Detail Record** is the description of a concluded charging session. The CDR is the only billing-relevant object. CDRs are sent from the CPO to the eMSP after the charging session has ended.

Although there is no requirement to send CDRs in near realtime, it is seen as good practice to send them as soon as possible. But if there is an agreement between parties to send them, for example, once a month, that is also allowed by OCPI.

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

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

8.1.1. Credit CDRs

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

When a CPO wants to make changes to a CDR that was already sent to the eMSP, the CPO has to send a Credit CDR for the first CDR. How to do this is described in the use case Send a Credit CDR.

8.1.2. Replication model

When the CPO creates CDR(s) they push them to the relevant eMSP with Party Issued Object replication. A CPO is not required nor expected to send *all* CDRs to *all* eMSPs. Typically a CPO will only send a certain CDR to the eMSP that the CDR is relevant to.

CDRs should contain enough information (dimensions) to allow the eMSP to validate the total cost.

If an eMSP cannot validate the cost of the CDR, or their cost calculation gives a different outcome from the cost given in the CDR object from the CPO, they can disupte the CDR with the Dispute a CDR use case.

Note that an eMSP might have a very different contract/pricing model with their EV drivers than the tariff structure of the CPO. The cost given in the CDR object is what the eMSP owes the CPO for the Charging Session.

8.1.3. Changes from OCPI 2.2.1

- The `country_code`, `party_id` and `id` fields were removed as they were from all Party Issued Objects. These fields are now given in the Party Issued Object Update object in the request that pushes the Tariff object.

- Added functionality of disputing a CDR.
- Added a specification of how an eMSP can check the cost of a CDR that the CPO computed.
- Added a decimal number type to use for costing-related numbers.
- Added `tariff_id` and `tariff_association_id` fields to the CDR object.
- Added a `token_version` field to the CdrToken object so CPO can inform eMSP of which version of the Token it used for whitelist authorization.

8.2. Replicating CDR objects

8.2.1. UC: 08.01 - Replicate CDR objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains a copy of the CDR objects that are issued to them by Party Y on a Platform B
2*	**Description**	Using the use cases of the Party-Issued Objects chapter, CDR objects are replicated from Party Y to Party X. Unlike most other Party Issued Objects, CDRs are never updated after being issued.
3	**Actors**	eMSP, CPO, Hub
4	**Flow**	1. Party X subscribes to Party Y's CDR objects 3. Party Y pushes every newly created CDR object to Party X as soon as these updates happen 4. This continues until either party cancels the subscription
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's CDRs to Party X on Platform A.
6	**Postconditions**	Party X has up-to-date information on the CDRs issued by Party Y
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 40. UC: 08.01 Requirements

ID	Precondition	Requirement
R.08.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "cdrs" as the ModuleID value.
R.08.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.08.01.01

ID	Precondition	Requirement
R.08.01.03	Platform B sends a Party Issued Update update to Party Y on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.08.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the CDR object type.

8.2.2. UC: 08.02 - Send a Credit CDR

1	**Objective(s)**	**1.** Party Y on Platform B informs Party X on Platform A that Party Y wants to credit a CDR that they issued to Party X earlier.
2	**Description**	This use case allows the CPO (Party Y) to inform the eMSP (Party X) that one of Party Y's CDRs was in error and Party Y would like to settle the Session differently. This is done by sending a second CDR object that references the credited CDR object and signals to Party X that, if possible, they should not bill the CDR, or undo any billing of it that they already did.
3	**Actors**	eMSP, CPO, Hub
4	**Flow**	1. Party Y on Platform B issues a credit CDR to Party X on Platform A using the Party Issued Object replication use cases.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Party X on Platform A is subscribed to the CDRs module of Party Y on Platform B. Party Y previously issued a CDR to Party X that they now wish to credit.
6	**Postconditions**	Party X knows Party Y credited one of Party Y's CDRs, and Party Y knows Party X received Party Y's credit CDR.
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

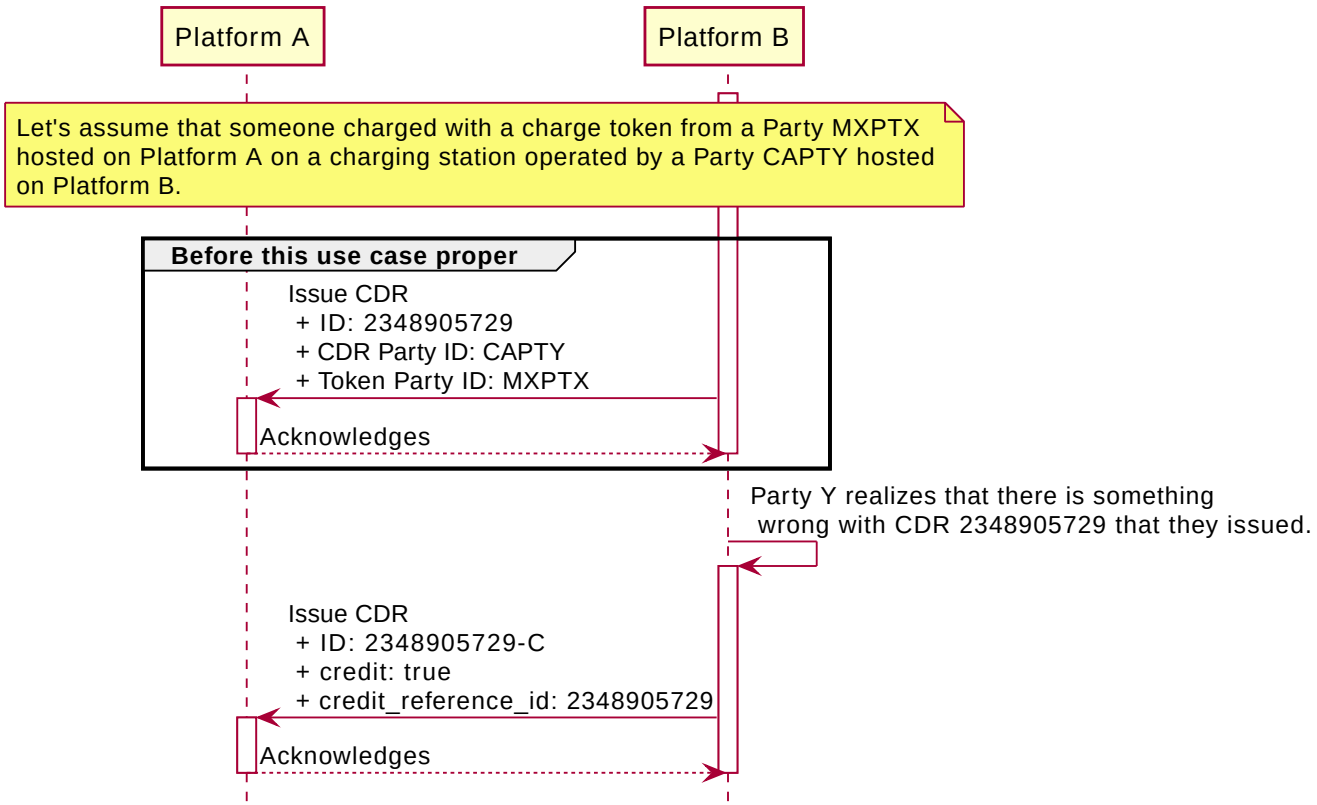


Figure 45. Sequence Diagram: Send a Credit CDR

Table 41. UC: 08.02 Requirements

ID	Precondition	Requirement
R.08.02.01		Platform B SHALL make the request to issue a credit CDR for Party Y according to the requirements listed for every CDR in Replicate a CDR.
R.08.02.02		Platform B SHALL set the `id` field of the PartyIssuedObjectUpdate object in their request to Platform A to something different from the value of the Party Issued Object ID of the credited CDR.
R.08.02.03		Platform B MAY set the `id` field of the PartyIssuedObjectUpdate object in their request to Platform A to the value of the Party Issued Object ID of the credited CDR with a string of up to three code points appended.
R.08.02.04		Platform B MAY set the `id` field of the PartyIssuedObjectUpdate object in their request to Platform A to a completely different number compared to the ID of the credited CDR.
R.08.02.05		Platform B SHALL set the `credit` field of the CDR object in their request to `true` to indicate that a CDR is a Credit CDR.

ID	Precondition	Requirement
R.08.02.06		The Credit CDR references the old CDR via the `credit_reference_id` field, which SHALL contain the `id` of the original CDR.
R.08.02.07		The values in the `total_cost` field SHALL contain the negative amounts of the credited CDR.
R.08.02.08		Except where requirements R.08.02.01 through R.08.02.07 require something else, the Credit CDR SHOULD contain all the same data as the original CDR object.
R.08.02.09		After having sent the Credit CDR, the CPO MAY send a new CDR for the same Session that the credited CDR was for, with a new unique ID, and with the fields `credit` and `credit_reference_id` omitted.

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

8.2.3. Example of a CDR


```
{
  "start_date_time": "2015-06-29T21:39:09Z",
  "end_date_time": "2015-06-29T23:37:32Z",
  "cdr_token": {
    "party_id": "DETNM",
    "uid": "012345678",
    "type": "RFID",
    "contract_id": "DE8ACC12E46L89"
  },
  "auth_method": "WHITELIST",
  "cdr_location": {
    "id": "LOC1",
    "name": "Gent Zuid",
    "address": {
      "address": "F.Rooseveltlaan 3A",
      "city": "Gent",
      "postal_code": "9000",
      "country": "BEL",
      "coordinates": {
        "latitude": "3.729944",
        "longitude": "51.047599"
      }
    },
    "evse_uid": "3256",
    "evse_id": "BE*BEC*E041503003",
    "connector_id": "1",
    "connector_standard": "IEC_62196_T2",
    "connector_format": "SOCKET",
    "connector_power_type": "AC_1_PHASE"
  },
  "currency": "EUR",
  "tariff_association_id": "TAC1",
  "tariff_id": "TAR1",
  "charging_periods": [{
    "start_date_time": "2015-06-29T21:39:09Z",
    "dimensions": [{
      "type": "TIME",
```

```
      "volume": 1.973
    }]
  }],
  "total_cost": {
    "before_taxes": 3.95,
    "taxes": [
      {
        "name": "VAT",
        "amount":  0.39
      }
    ]
  },
  "total_energy": 15.342,
  "total_time": 1.973,
  "total_time_cost": {
    "before_taxes": 3.95,
    "taxes": [
      {
        "name": "VAT",
        "amount": 0.39
      }
    ]
  },
  "last_updated": "2015-06-29T22:01:13Z"
}
```

8.3. Remote Procedure Calls on CDR Objects

8.3.1. UC: 08.03 - Dispute a CDR

1	**Objective(s)**	**1.** Party X on Platform A notifies Party Y on Platform B that Party X disputes a CDR that Party Y issued to Party X.
2	**Description**	This use case allows the eMSP (Party X) to inform the CPO (Party Y) that they think there is some sort of error with a CDR that Party Y issued to Party X.
3	**Actors**	eMSP, CPO, Hub
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the identifier of the CDR that Party X disputes.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Party Y issued a CDR to Party X.
6	**Postconditions**	Party Y knows that Party X disputes a CDR of theirs.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

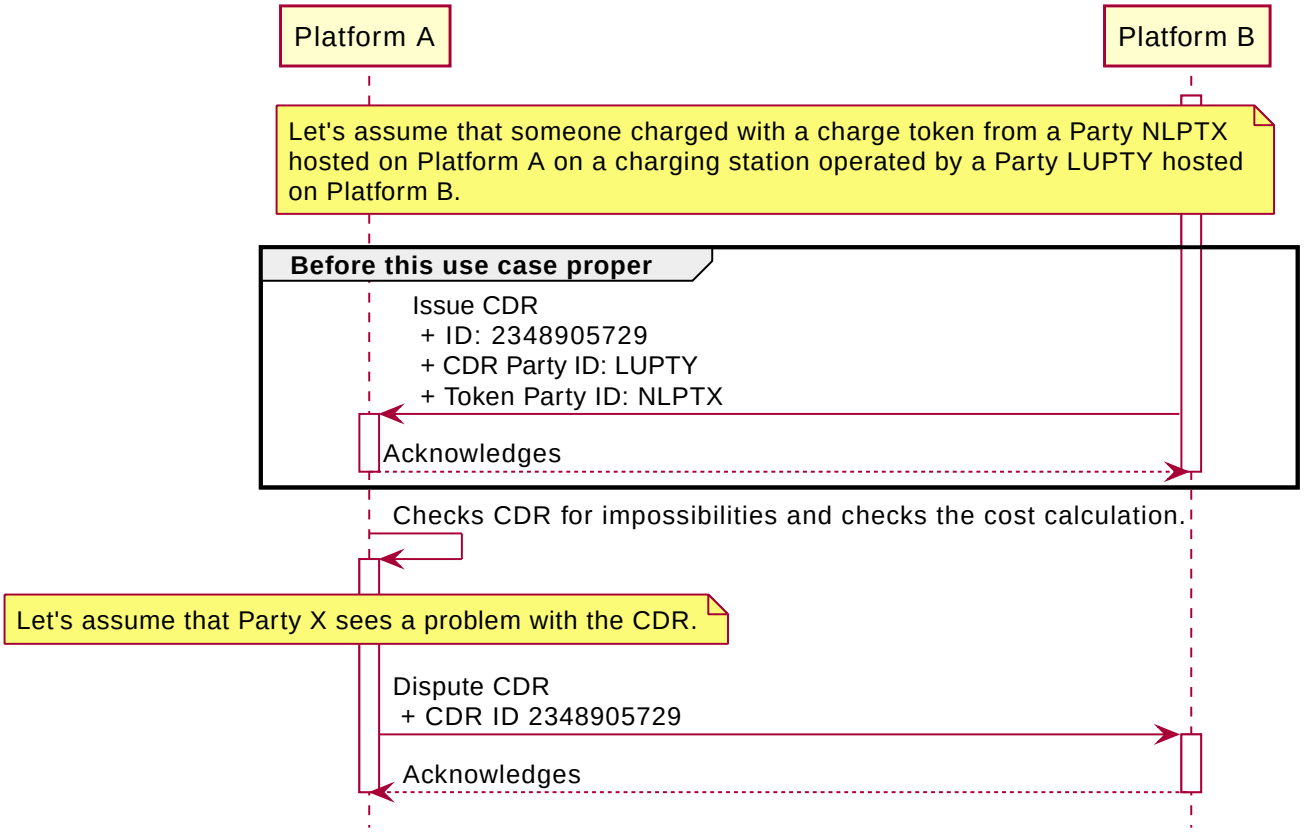


Figure 46. Sequence Diagram: Dispute a CDR

Table 42. UC: 08.03 Requirements

ID	Precondition	Requirement
R.08.03.01		Platform A SHALL make the request to dispute a CDR following Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.08.03.02		Platform A SHALL use "dispute" as the operation name for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.08.03.03		Platform A SHALL use POST as the HTTP request verb when making its request.
R.08.03.04		Platform A SHALL use a DisputeCdrRequest in the payload field of the request body for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.08.03.05		Platform B SHALL include a DisputeCdrResponse object in the payload field of the OcpiResponse object in the response body according to Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.

8.4. Other CDRs use cases

8.4.1. UC: 08.04 - Check CDR price

1	**Objective(s)**	**1.** Party X on Platform A confirms that the price given in a CDR or a Session that they received from Party Y on Platform B is correct according to Tariffs that Party Y previously announced to Party X.
2	**Description**	When a Party receives a CDR or a Session they will typically want to check if the price that they are charged for it is correct. This use case explains how to do that.
3	**Actors**	eMSP
4	**Flow**	This use case happens entirely within the eMSP's systems.
5	**Preconditions**	Party X hosted on Platform A is subscribed to Tariffs of Party Y hosted on Platform B. Party X is subscribed to Tariff Associations of Party Y. Party X is subscribed to Sessions and/or CDRs of Party Y. Party X received a CDR or Session object from Party Y that it wants to check Party Y's cost computation for.
6	**Postconditions**	Party X knows what the price of the CDR or the Session should have been according to Tariff information given to Party X previously by Party Y.
7	**Error handling**	None specified.
8	**Remark(s)**	Note that OCPI does not facilitate checking the physical quantities that the CPO claims to have delivered. That the CPO measures time and energy accurately can only be guaranteed by checking that they use properly calibrated measurement equipment and that the data from such equipment is not altered in transit. To enable the eMSP to verify these things remotely, one would need a scheme like the German Eichrecht's rules for remote transmission of measurements. In absence of widespread enactment of such rules, the eMSP will have to trust the CPO to comply with applicable metrology laws and accordingly use decent calibrated meters and not tamper with the data in transit. What OCPI can provide, and does provide with this use case, is checking that the price is calculated correctly given the trusted measurements. Also OCPI provides the `calibration_info_url` field in the EVSE data type which allows eMSPs or their Drivers to check calibration evidence manually.

Table 43. UC: 08.04 Requirements

ID	Precondition	Requirement
R.08.04.01		Party X SHOULD start checking the cost by finding which Tariff Association applies to the CDR or Session. To do so, it selects from all Tariff Associations replicated to it by Party Y on Platform B, all those that fulfill all of these conditions: * The Tariff Association has the pair of `evse_uid` and `connector_id` for the connector that the Session to check happened on in their `connectors` property; * The Tariff Association has a `start_date_time` that is earlier than or equal to the `start_date_time` of the Session to check; * The Tariff Association has an `audience` that matches the `default_profile_type` of the Token that the Session was started with, or that equals `REGULAR` if that Token has no `default_profile_type` set. Out of all the Tariff Associations yielded by the selection, it picks the one that has the latest value for `start_date_time`. If there are multiple Tariff Associations that have the same value for `start_date_time`, it picks the one it received last out of them.
R.08.04.02	Party X found no Tariff Association object matching the criteria listed in R.08.04.01.	Party X SHOULD consider the check failed. Party X MAY Dispute the CDR with Party Y if Party X is checking a CDR object.
R.08.04.03	Party X found a Tariff Association object matching the criteria listed in R.08.04.01.	Party X SHOULD check if the Tariff Association ID in the `tariff_association_id` field of the Session or CDR object is the same as the ID of the Tariff Association selected according to R.08.04.01.
R.08.04.04	Party X found a Tariff Association object matching the criteria listed in R.08.04.01.	Party X SHOULD check if the Tariff ID in the `tariff_id` field of the Session or CDR object is the same as the value of the `tariff_id` field in the Tariff Association selected according to R.08.04.01.
R.08.04.05	Party X found a matching Tariff Association object when executing R.08.04.01 and Party X found that the checks of R.08.04.03 and R.08.04.04 succeded.	Having established which Tariff Association applies to the Session or CDR according to R.08.04.01, Party X SHOULD load the Tariff that has the Party Issued Object ID given in the `tariff_id` field of that Tariff Association and that was replicated to Party X from Party Y and Platform B.
R.08.04.06	Party X found no Tariff when executing 08.04.05	Party X SHOULD consider the check failed. Party X MAY Dispute the CDR with Party Y if Party X is checking a CDR object.
R.08.04.07	Party X found a Tariff when executing R.08.04.05	Party X SHOULD check if the currency in the `currency` field of the Session or CDR object is the same as that of the Tariff selected according to R.08.04.03.

ID	Precondition	Requirement
R.08.04.08	The tariff selected according to R.08.04.05 passes the check of R.08.04.07.	Party X SHOULD compute the cost of the Session or CDR, according to the Tariff object format descriptions in the Tariffs and CDRs chapters of this specification. Party X SHOULD use as inputs to this computation only these data: * The Tariff selected at R.08.04.05; * The facts of the charging Session as given in the `charging_periods`, `start_date_time` and `end_date_time` fields of the Session or CDR object from Party Y; * Whether the object to check the cost for is a credit CDR, as given in the `credit` field if it is a CDR object; and * Whether the object describes a home charging session, as given in the `home_charging_compensation` field if the object is a CDR.
R.08.04.09	Party X computed a cost according to R.08.04.08	Party X SHOULD check if the costs with and without taxes as determined while executing R.08.04.08 match those given by Party Y in the object's `total_cost` field. Party Y MAY allow for some error margin to allow small differences in rounding behavior.
R.08.04.10	The checks done by Party X according to R.08.04.03, R.08.04.04, R.08.04.07 or R.08.04.09 fail.	Party X SHOULD consider the check failed. Party X MAY Dispute the CDR with Party Y if Party X is checking a CDR object.

8.5. Data types

8.5.1. AuthMethod *enum*

Value	Description
AUTH_REQUEST	Authorization request has been sent to the eMSP.
COMMAND	OCPI commands like Start a Session or Reserve an EVSE at a Location were used to start the Session, with the Token provided in the command being used as authorization.
WHITELIST	Whitelist was used for autorization, no request to the eMSP was performed.

8.5.2. CDR *class*

The *CDR* object describes the Charging Session and its costs, and how these costs are derived from the recorded facts of the Charging Session.

The CDR object is different from the Session object. The Session object is dynamic as it reflects the current state of the Charging Session while the Charging Session is ongoing. The information is meant to be presented to the Driver while the Charging Session is ongoing.

The CDR on the other hand can be thought of as *sealed*, preserving the information valid at the moment in time the underlying Session was started. This is a requirement of the main use case for CDRs, namely invoicing. If e.g. a street is renamed the day after a session took place, the driver should be presented with the name valid at the time the

Session was started. This guarantees that the CDR will be recognized as correct by the Driver and is not going to be disputed.

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

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

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

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

Property	Type	Card.	Description
start_date_time	DateTime	1	Start timestamp of the charging session, or in-case of a reservation (before the start of a session) the start of the reservation.
end_date_time	DateTime	1	The timestamp when the session was completed/finished, charging might have finished before the session ends, for example: EV is full, but parking cost also has to be paid.
session_id	CiString[1..36]	?	Unique ID of the Session for which this CDR is sent. Is only allowed to be omitted when the CPO has not implemented the Sessions module or this CDR is the result of a reservation that never became a charging session, thus no OCPI Session.
cdr_token	CdrToken	1	Token used to start this charging session, including all the relevant information to identify the unique token.
auth_method	AuthMethod	1	Method used for authorization. Multiple AuthMethods are possible during a charging sessions, for example when the session was started with a reservation: Reserve an EVSE at a Location: `COMMAND`. When the driver arrives and starts charging using a Token that is whitelisted: `WHITELIST`. The last method SHALL be used in the CDR.

Property	Type	Card.	Description
authorization_reference	CiAsciiString[1..36]	?	Reference to the authorization given by the eMSP. When the eMSP provided an `authorization_reference` in either: real-time authorization, StartSession or ReserveNow, this field SHALL contain the same value. When different `authorization_reference` values have been given by the eMSP that are relevant to this Session, the last given value SHALL be used here.
cdr_location	CdrLocation	1	Location where the charging session took place, including only the relevant EVSE and Connector.
meter_id	AsciiString[1..255]	?	Identification of the Meter inside the Charge Point.
currency	CiAsciiString[3]	1	Currency of the CDR in ISO 4217 Code.
tariff_association_id	Tariff	1	The ID of the Tariff Association that was used to look up the Tariff of this CDR. When the session is free, the ID of a Tariff Association for a *Free of Charge* tariff is to be given in this field.
tariff_id	Tariff	1	The ID of the Tariff that was used to compute what the Session of this CDR costs. When the session is free, the ID of a *Free of Charge* tariff is to be given in this field.
charging_periods	ChargingPeriod	+	List of Charging Periods that make up this charging session.
signed_data	SignedData	?	Signed data that belongs to this charging Session.
total_cost	Price	1	Total sum of all the costs of this transaction in the specified currency.
total_fixed_cost	Price	?	Total sum of all the fixed costs in the specified currency, except fixed price components of parking and reservation. The cost not depending on amount of time/energy used etc. Can contain costs like a start fee.
total_energy	decimal	1	Total energy charged, in kWh.
total_energy_cost	Price	?	Total sum of all the cost of all the energy used, in the specified currency.
total_time	decimal	1	Total duration of the charging session in hours.
total_time_cost	Price	?	Total sum of all the cost related to duration of charging during this transaction, in the specified currency.
total_reservation_cost	Price	?	Total sum of all the cost related to a reservation of a Charge Point, including fixed price components, in the specified currency.
remark	UnicodeString[1..255]	?	Optional remark, can be used to provide additional human readable information to the CDR, for example: reason why a transaction was stopped.

Property	Type	Card.	Description
credit	boolean	?	When set to `true`, this is a Credit CDR, and the field `credit_reference_id` needs to be set as well.
credit_reference_id	CiAsciiString[1..39]	?	Is required to be set for a Credit CDR. This SHALL contain the `id` of the CDR for which this is a Credit CDR.
home_charging_compensation	boolean	?	When set to `true`, this CDR is for a charging session using the home charger of the EV Driver for which the energy cost needs to be financial compensated to the EV Driver.
last_updated	DateTime	1	Timestamp when this CDR was last updated (or created).

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

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

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

8.5.3. CdrConnector *class*

A reference to the connector that the Session described in the CDR happened on, plus some convenient data about that connector.

evse_uid	**CiAsciiString[1..36]**	**1**	**Uniquely identifies the EVSE among all EVSEs of all Locations of the same Party. Refers to the `uid` field of the EVSE objects in Locations issued to the Party receiving the CDR.**
evse_id	CiAsciiString[1..48]	1	Compliant with the following specification for EVSE ID from "eMI3 standard version V1.0" (http://emi3group.com/documents-links/) "Part 2: business objects".
connector_id	CiAsciiString[1..36]	1	Identifier of the connector within the EVSE.
connector_standard	ConnectorType	1	The standard of the connector that the Session described by the CDR happened on.
connector_format	ConnectorFormat	1	The format (socket/cable) of the connector that the Session described by the CDR happened on.
connector_power_type	PowerType	1	The power type of the connector that the Session described by the CDR happened on.

8.5.4. CdrDimension *class*

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

8.5.5. CdrDimensionType *enum*

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

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

NOTE

OCPI makes it possible to provide SoC in the Session object. This information can be useful to show the current State of Charge to an EV driver during charging. Implementers should be aware that SoC is only available at some DC Chargers. Which is currently a small amount of the total amount of Charge Points. Of these DC Chargers, only a small percentage currently provides SoC via OCPP to the

CPO. Then there is also the question if SoC is allowed to be provided to third-parties as it can be seen as privacy-sensitive information. So if an implementer wants to show SoC in, for example an App, care should be taken, to make the App work without SoC, as this will probably not always be available.

8.5.6. CdrLocation *class*

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

Property	Type	Card.	Description
id	CiAsciiString[1..36]	1	Party Issued Object identifier of the Location. Uniquely identifies the Location among all Locations issued by the same Party on the same Platform. This field can never be changed, modified or renamed.
name	UnicodeString[1..255]	?	Display name of the Location.
address	Address	?	Street address and geographical coordinates of the Loction where the Session happened. While the field is optional to allow for "privacy by design" solutions where no address data of a home Charging Station is ever shared, it must typically be filled in for the usual case of a transaction on a Charging Station that is open to the general public. That is, CDR senders should only leave this field unset if they agreed with the receiver that it is OK to send them CDRs without an address.
connector	CdrConnector	?	A reference to the connector that the Session happened on. This is allowed to be unset if and only if this CDR was created for a reservation that never resulted in a Charging Session.

8.5.7. CdrToken *class*

Property	Type	Card.	Description
party_id	CiAsciiString[5]	1	ISO 15118 party ID of the Party that issued this Token.
uid	CiAsciiString[1..36]	1	Unique ID by which this Token can be identified. This is the field used by the CPO's system (RFID reader on the Charge Point) to identify this token. Currently, in most cases: `type=RFID`, this is the RFID hidden ID as read by the RFID reader, but that is not a requirement. If this is a `type=APP_USER` Token, it will be a unique, by the eMSP, generated ID.
type	TokenType	1	Type of the token
contract_id	CiAsciiString[1..36]	1	Uniquely identifies the EV driver contract token within the eMSP's platform (and suboperator platforms). Recommended to follow the specification for eMA ID from "eMI3 standard version V1.0" (http://emi3group.com/documents-links/) "Part 2: business objects."

Property	Type	Card.	Description
token_version	int	?	If whitelist authorization was used by the CPO to authorize the Session, this field MUST be filled with the Party Issued Object version of the Token that was used by the CPO to authorize the Session.

8.5.8. ChargingPeriod *class*

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

Property	Type	Card.	Description
start_date_time	DateTime	1	Start timestamp of the charging period. A period ends when the next period starts. The last period ends when the session ends.
dimensions	CdrDimension	+	List of relevant values for this charging period.

8.5.9. DisputeCdrRequest *class*

Property	Type	Card.	Description
cdr_id	CiAsciiString[1..36]	1	The Party Issued Object ID of the CDR that is being disputed.
reason	UnicodeString[1..4000]	1	An explanation of why the CDR is disputed. This can be a human-written note or automatically generated text from an automated checking system.

8.5.10. DisputeCdrResponse *class*

This object type has no fields.

8.5.11. SignedData *class*

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

Property	Type	Card.	Description
encoding_method	CiAsciiString[1..36]	1	The name of the encoding used in the SignedData field. This is the name given to the encoding by a company or group of companies. See note below.
encoding_method_version	int	?	Version of the EncodingMethod (when applicable)
public_key	AsciiString[1..512]	?	Public key used to sign the data, base64 encoded.
signed_values	SignedValue	+	One or more signed values.

Property	Type	Card.	Description
url	AsciiString[1..512]	?	URL that can be shown to an EV driver. This URL gives the EV driver the possibility to check the signed data from a charging session.

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

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

8.5.12. SignedValue *class*

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

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

9. Tariffs

This chapter describes the Tariffs module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the Tariff module, the type of data object is the Tariff object. A Tariff object describes how a cost can be computed for a Session or CDR object.

The main purposes of the Tariffs module are the following:

- It allows CPOs to announce to MSPs how they charge for Charging Sessions on their Locations.
- It allows MSPs, NAPs and NSPs to automatically show tariff information to Drivers in for example smartphone apps.
- It allows MSPs to validate the cost that CPOs charge them for Charging Sessions.

9.1. Changes from OCPI 2.2.1

- A Tariffs Associations module was split off from Tariffs. After the split, a Tariff only serves to describe a way in which a cost can be computed for any Session or CDR. A Tariff no longer describes to which Sessions and CDRs it applies, as OCPI 2.2.1 did using the start date, end date and type fields. Associating Tariffs with Sessions is now the responsibility of the Tariff Associations module.
- The `country_code`, `party_id` and `id` fields were removed as they were from all Party Issued Objects. These fields are now given in the Party Issued Object Update object in the request that pushes the Tariff object.
- The `step_size` field was removed from the PriceComponent object. In OCPI 3.0, CPOs are expected to measure every quantity with at least the minimum precision required by applicable metrology law and carry this precision through in further cost computation. See the notes for the decimal type for pointers on how to implement this in code.
- It is now clear when `FLAT` Tariff components apply.
- The `PARKING_TIME` dimension has been removed. Both time spent charging and time spent not charging are now counted under the dimension `TIME`, and restrictions on TariffElements have to be used to charge a different price for time spent charging than for time spent not charging.

9.2. A note on "Parking time", "Loitering fees", "Idle penalties", et cetera

Many CPOs want to restrict the time that vehicles are occupying EVSEs without charging. This makes sense as charging stations are an expensive investment, and to make the best use of these charging stations, the CPO or their stakeholders will want the charging stations to be charging vehicles as much as possible.

To incentivize Drivers to vacate an EVSE quickly after the charging of their vehicle is completed, many CPOs have begun to charge extra for time that a vehicle is attached to the EVSE but not charging. OCPI 2.2.1 had a specific dimension *PARKING_TIME* to facilitate such surcharges for time in a Charge Session spent not charging. We will call such surcharges *loitering fees* for the rest of this section, although they are also known as *idle fees* or *parking fees* or other such terms.

In the OCPI 3.0 development process, we noticed several problems with such loitering fees:

- It is hard for a CPO to know precisely when energy transfer starts or ends. Especially with OCPP 1.6 and earlier OCPP versions, there is no reliable way to know when exactly charging ends or begins. Meanwhile, the starting point, endpoint and total energy consumption of a Charge Session are reliably captured by OCPP 1.6 charging stations.
- Even if the CPO knows when energy transfer stops, it is not always clear if this cessation of energy transfer is caused by the vehicle, by the charging station or by other circumstances in the local electrical system. If the cause is not in the vehicle, the CPO would be imposing surcharges on the Driver that the Driver cannot avoid. This is intransparent and unfair to the Driver and may also be a compliance risk for the CPO.
- Even if the CPO implements loitering fees correctly, with solutions to the problems above, then a Driver can work around the CPO's incentives by programming the car. The Driver can program the car to never let the charging power drop to zero, for example by making the car charge at full speed until the battery is 75% full and then making the car charge the last 25% as slow as possible, putting off the loitering fee as long as possible.
- It is unclear if and how a loitering fee would be applied if a vehicle repeatedly stops and resumes charging during one Session.

For these reasons, the authors of OCPI believe a CPO should think twice before deciding on a loitering fee as the means for incentivizing Drivers to vacate EVSEs when charging is done. We believe CPOs should also consider other incentives, like for example:

- Charging a higher rate per hour after a certain duration of the Charging Session, regardless of when charging ends. This is a less precise tool than a loitering fee, but it is easier to understand for Drivers and easier to enforce for CPOs.
- Announcing maximum EVSE occupation durations with signage and letting on-site staff enforce this.
- Designing the site layout to encourage quick turnaround. This is how fuel stations seem to get by without loitering fees nor signs setting maximum occupation durations.

The example Tariffs below frequently give examples of how to set loitering fees with the `vehicle_requesting_power` TariffRestriction. This is done because such Tariffs are in common use, not because we as authors believe that such Tariffs are typically the best choice for CPOs.

9.3. Replicating Tariff objects

9.3.1. UC: 09.01 - Replicate Tariff objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains and maintains up-to-date copies of the Tariffs used by Party Y
2	**Description**	Using the use cases of the Party-Issued Objects chapter, Tariff objects are replicated from Party Y to Party X
3	**Actors**	eMSP, CPO, NAP, NSP
4	**Flow**	1. Party X subscribes to Party Y's Tariff objects 2. Party Y pushes all their Tariff objects as of subscription time to Party X 3. Party Y pushes every newly updated Tariff object to Party X as soon as these updates happen 4. This continues until either party cancels the subscription

5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Tariffs module to Party X on Platform A.
6	**Postconditions**	Party X has up-to-date information on the Tariff objects of Party Y
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 44. UC: 09.01 Requirements

ID	Precondition	Requirement
R.09.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "tariffs" as the ModuleID value.
R.09.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.09.01.01
R.09.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.09.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the Tariff object type.
R.09.01.04		Tariff objects are immutable. That is, Platform B SHALL NOT send an update with a changed version number for a previously received object using use case Send a Full Update of a Party-Issued Object to a Subscribed Platform in the context of a subscription to the Tariffs module.
R.09.01.05	Platform A receives an update with a changed version number for a previously received object using use case Send a Full Update of a Party-Issued Object to a Subscribed Platform	Platform A SHALL respond with an OcpiResponse with no value for the `payload` field and with the `status_code` field set to 6003.

9.3.2. Examples of Tariff objects

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

9.3.2.1. Simple Tariff example € 0.25 per kWh

- Energy
  - € 0.25 per kWh (excl. VAT)

- 10% VAT

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

```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2018-12-17T11:16:55Z"
}
```

9.3.2.2. Simple Tariff example with British Columbia taxes

In some jurisdictions, there is no single national VAT scheme and multiple taxes may apply to the cost of a Charging Session. This is an example of such a situation in the Canadian province of British Columbia.

- Time
  - C$ 1.00 per hour (before taxes)
  - 5% GST (Goods and Services Tax imposed by the Canadian federal government)
  - 7% PST (Provincial Sales Tax imposed by the province of British Columbia)

This Tariff will results in costs of C$ 2.00 (before taxes) or C$ 2.24 (after taxes) when a Driver is in a Charging Session for 2 hours.

```
{
  "currency": "CAD",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 1.00,
      "taxes": [
        {
          "name": "GST",
          "percentage": 5
        },
        {
          "name": "PST",
          "percentage": 7
        }
      ]
    }]
  }],
  "last_updated": "2024-02-16T18:42:55Z"
}
```

9.3.2.3. Tariff example € 0.25 per kWh + start fee

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

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

```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2018-12-17T11:36:01Z"
}
```

9.3.2.4. Tariff example € 0.25 per kWh + minimum price

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

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

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

```
{
  "currency": "EUR",
```

```
  "min_price": {
    "before_taxes": 0.50,
    "taxes": [{
      "name": "VAT",
      "amount": 0.05
    }]
  },
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2018-12-17T16:45:21Z"
}
```

9.3.2.5. Tariff example € 0.25 per kWh + loitering fee + start fee

- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT
- Time occupying the EVSE while not requesting power
  - € 2.00 per hour (excl. VAT)
  - 20% VAT

For a Charging Session where 20 kWh are charged and the Session keeps going for 40 minutes after the charging was completed, this tariff will result in costs of € 6.83 (excl. VAT) or € 7.70 (incl. VAT).

```
{
  "currency": "EUR",
  "elements": [
    {
      "price_components": [{
        "type": "TIME",
        "price": 2.00,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 20
          }
        ]
      }],
      "restrictions": {
        "vehicle_requesting_power": false
      }
    },
    {
      "price_components": [{
        "type": "FLAT",
```

```
        "price": 0.50,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 20
          }
        ]
      }, {
        "type": "ENERGY",
        "price": 0.25,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 10
          }
        ]
      }]
    }
  ],
  "last_updated": "2018-12-17T11:44:10Z"
}
```

9.3.2.6. Tariff example € 0.25 per kWh + start fee + max price

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

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

```
{
  "currency": "EUR",
  "max_price": {
    "before_taxes": 10.00,
    "taxes": [{
      "name": "VAT",
      "amount": 1.00
    }]
  },
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }, {
```

```
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2018-12-17T17:15:01Z"
}
```

9.3.2.7. Simple Tariff example € 2 per hour

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

- Charging Time
  - € 2.00 per hour (excl. VAT)
  - 10% VAT

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

Note that the Session lasts the whole time that the vehicle is known to the CPO to be occupying the EVSE, so with this Tariff, what the Driver pays is not dependent on how much time of those 2.5 hours they spend charging or not charging.

```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

9.3.2.8. Simple Tariff example € 3 per hour, € 5 per hour loitering

Example of a tariff where the driver pays for the time of using the EVSE, but pays more when the car is no longer charging, to discourage the Driver from leaving their vehicle connected when it is already full.

- Time (when not drawing power)
  - € 5.00 per hour (excl. VAT)
  - 20% VAT
- Time (otherwise)
  - € 3.00 per hour (excl. VAT)

- 10% VAT

A charging session of 2.5 hours of charging, where the vehicle is occupying the EVSE for 42 more minutes after charging ended, results in a total session time of 150 minutes (charging) + 42 minutes (not charging). This session with this tariff will result in total cost of € 11.00 (excl. VAT) or € 12.45 (incl. VAT).

```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 5.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }],
    "restrictions": {
      "vehicle_requesting_power": false
    }
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 3.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2018-12-17T17:00:43Z"
}
```

9.3.2.9. Simple Tariff example with multiple languages

- Time
  - € 1.90 per hour (excl. VAT)
  - 5.2% VAT

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

```
{
  "currency": "EUR",
  "tariff_alt_text": [{
    "language": "en",
    "text": "€ 2.00 per hour including VAT"
  }, {
    "language": "nl",
    "text": "€ 2,00 per uur inclusief BTW"
  }],
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 1.90,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 5.2
```

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

9.3.2.10. Tariff example not possible with OCPI: differentiation by payment method

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

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

```
{
  "currency": "EUR",
  "tariff_alt_text": [{
    "language": "en",
    "text": "€ 2.00 per hour, plus a flat fee of € 0.25 for debit cards or € 0.50 for credit cards. These prices
include VAT."
  }, {
    "language": "nl",
    "text": "€ 2,00 euro per uur, plus een starttarief van € 0,25 met bankpas of € 0,50 euro met creditcard. Deze
prijzen zijn inclusief BTW."
  }],
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.40,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 25
        }
      ]
    }, {
      "type": "TIME",
      "price": 1.90,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 5.2
        }
      ]
    }]
  }],
  "last_updated": "2018-12-29T15:55:58Z"
}
```

9.3.2.11. Simple Tariff example with alternative URL

This examples shows the use of `tariff_alt_url`.

This shows a tariff where the price might not be fixed, but depend on the real-time energy prices. To explain this to the driver, a short text inside `tariff_alt_text` might not be the best solution. Showing a graph could be better. Therefore it is also possible to provide an URL in `tariff_alt_url` to a site that explains the tariff better and in more detail.

NOTE	While it is possible in OCPI to refer to a URL for pricing, consumer law may place restrictions on the

> variability of tariffs and the indirections that Drivers have to go through to learn about these tariffs. In general, it is intransparent to consumers to Drivers to report one tariff via OCPI but also use a URL or alt text to override the information in the Tariff object. Use such an approach with caution.

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

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

- Start fee: € 0.50 (excl. VAT), € 0.60 (incl. VAT)
- Energy costs: € 5.11 (excl. VAT), € 5.62 (incl. VAT)
- Total: € 5.61 (excl. VAT), € 6.22 (incl. VAT)

if the announced prices were billed.

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

- Start fee: € 0.50 (excl. VAT), € 0.60 (incl. VAT)
- Energy costs: € 4.50 (excl. VAT), € 4.95 (incl. VAT)
- Total: € 5.00 (excl. VAT), € 5.55 (incl. VAT)

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

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

```
{
  "currency": "EUR",
  "tariff_alt_url": "https://company.com/tariffs/13",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
```

```
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

9.3.2.12. Complex Tariff example

- Start or transaction fee
  - € 2.50 (excl. VAT)
  - 15% VAT
- Time
  - When the vehicle is not requesting power, on weekdays between 09:00 and 18:00
    - € 5 per hour (excl. VAT)
    - 10% VAT
  - When the vehicle is not requesting power, on Saturday between 10:00 and 17:00
    - € 6 per hour (excl. VAT)
    - 10% VAT
  - When charging with less than 32A
    - € 1.00 per hour (excl. VAT)
    - 20% VAT
  - When charging with more than 32A on weekdays
    - € 2.00 per hour (excl. VAT)
    - 20% VAT
  - When charging with more than 32A on weekends
    - € 1.25 per hour (excl. VAT)
    - 20% VAT

For a charging session on a Monday morning starting at 09:30 where the charging takes 2:45 hours (165 minutes), and where the driver uses a maximum of 16A of current and is not requesting power for additional 42 minutes afterwards, this tariff will result in costs of € 8.75 (excl. VAT) or € 10.03 (incl. VAT) for a total session time of 165 minutes (charging) + 42 minutes (not charging).

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	2.50	2.50	15%	2.875

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time drawing power	165 minutes	1.00 per hour	2.75	20%	3.30
Time not drawing power	42 minutes	5.00 per hour	3.50	10%	3.85
Total			8.75		10.03

For a charging session on a Saturday afternoon starting at 13:30 which takes 1:54 hours (114 minutes), where the driver uses a minimum of 43A of current (all the time, which is only theoretically possible) and is loitering for additional 71 minutes afterwards, this tariff will result in a total cost of € 11.88 (excl. VAT) or € 13.42 (incl. VAT). A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	2.50	2.50	15%	2.875
Time drawing power	114 minutes	1.25 per hour	2.28	20%	2.736
Time not drawing power	71 minutes	6.00 per hour	7.10	10%	7.81
Total			11.88		13.421

```
{
  "currency": "EUR",
  "tariff_alt_url": "https://company.com/tariffs/14",
  "elements": [
    {
      "price_components": [{
        "type": "FLAT",
        "price": 2.50,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 15
          }
        ]
      }]
    }, {
      "price_components": [{
        "type": "TIME",
        "price": 5.00,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 10
          }
        ]
      }],
      "restrictions": {
        "start_time": "09:00",
        "end_time": "18:00",
        "day_of_week": ["MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"],
        "vehicle_requesting_power": false
      }
    }, {
      "price_components": [{
        "type": "TIME",
        "price": 6.00,
        "taxes": [
          {
```

```
            "name": "VAT",
            "percentage": 10
          }
        ]
      }],
      "restrictions": {
        "start_time": "10:00",
       "end_time": "17:00",
        "day_of_week": ["SATURDAY"],
        "vehicle_requesting_power": false
      }
    }, {
      "price_components": [{
        "type": "TIME",
        "price": 1.00,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 20
          }
        ]
      }],
      "restrictions": {
        "max_current": 32.00
      }
    }, {
      "price_components": [{
        "type": "TIME",
        "price": 2.00,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 20
          }
        ]
      }],
      "restrictions": {
        "min_current": 32.00,
        "day_of_week": ["MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"]
      }
    }, {
      "price_components": [{
        "type": "TIME",
        "price": 1.25,
        "taxes": [
          {
            "name": "VAT",
            "percentage": 20
          }
        ]
      }],
      "restrictions": {
        "min_current": 32.00,
        "day_of_week": ["SATURDAY", "SUNDAY"]
      }
    }
  ],
  "last_updated": "2015-06-29T20:39:09Z"
}
```

9.3.2.13. Free of Charge Tariff example

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

```
{
```

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

9.3.2.14. First hour free energy example

In this example, we have the following scenario:

- The first hour of loitering time is free.
- From the second to the fourth hour, loitering costs € 2.00 per hour
- From the fourth hour on, loitering costs € 3.00 per hour.
- The first kWh of energy is free, every additional kWh costs € 0.20.

Translated into our tariff schema, the pricing model looks like this:

- Energy
  - First kWh: free
  - Any additional energy
    - € 0.20 per kWh (excl. VAT)
- Time
  - When vehicle is requesting power: free
  - First hour of not requesting power: free
  - Second to fourth hours of not requesting power
    - € 2.00 per hour (excl. VAT)
  - Any more time than four hours in the Charge Session not requesting power
    - € 3.00 per hour (excl. VAT)

For a charging session where the driver charges 20 kWh and where the vehicle is loitering for 2:45 more hours after charging ended, this tariff will result in costs of € 7.30 (excl. VAT).

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT
Energy	1 kWh	0.00 per kWh	0.00
Energy	19 kWh	0.20 per kWh	3.80
Time not drawing power	60 minutes	0.00 per hour	0.00
Time not drawing power	105 minutes	2.00 per hour	3.50
Total			7.30

As no VAT information is given, it is not possible to calculate total costs including VAT.

Notice how in this Tariff, the order of TariffElements is used to make the switch to the higher rate for a longer time of not charging happen. The higher rate is given first in the array of TariffElements, so it has a higher priority. As long as the 3 hours of not drawing power have not elapsed, the restrictions for the first TariffElement in the list are not fulfilled, and so the second is tried. If 1 hour of not drawing power has not yet elapsed, the restrictions of the second TariffElement are not fulfilled either and no PriceComponent to charge for TIME is found at all. As a result TIME is free if the vehicle has not been loitering for an hour yet. If the vehicle has been loitering for at least an hour but not yet for three hours, the second TariffElement in the list is the first whose restrictions are fulfilled, and the rate for the TIME dimension from that element is used.

```
{
    "currency": "EUR",
    "elements": [
      {
        "price_components": [{
            "type": "TIME",
            "price": 3.0
        }],
        "restrictions": {
            "min_restrictions_duration": 10800,
            "vehicle_drawing_power": false
        }
      }, {
        "price_components": [{
            "type": "TIME",
            "price": 2.0
        }],
        "restrictions": {
            "min_restrictions_duration": 3600,
            "vehicle_drawing_power": false
        }
      }, {
        "price_components": [{
            "type": "ENERGY",
            "price": 0.0
        }],
        "restrictions": {
            "max_energy": 1.0
        }
      }, {
        "price_components": [{
            "type": "ENERGY",
            "price": 0.2
        }],
        "restrictions": {
            "min_energy": 1.0
        }
      }
    ],
    "last_updated": "2018-12-29T15:55:58Z"
}
```

9.3.2.15. Tariff example with reservation price

- Reservation
  - € 5.00 per hour (excl. VAT)
  - 20% VAT
- Start or transaction fee

    - € 0.50 (excl. VAT)
    - 20% VAT
- Energy
    - € 0.25 per kWh (excl. VAT)
    - 10% VAT

For a charging session that was started 15 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 6.75 (excl. VAT) or € 7.60 (incl. VAT).

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Reservation	15 minutes	5.00 per hour	1.25	20%	1.50
Total			6.75		7.60


```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 5.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
```

```
}
```

9.3.2.16. Tariff example with reservation price and fee

- Reservation
  - € 2.00 reservation fee (excl. VAT)
  - € 5.00 per hour (excl. VAT)
  - 20% VAT
- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT

For a charging session that was started 13 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 8.59 (excl. VAT) or € 9.80 (incl. VAT).

A breakdown is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	2.00	2.00	20%	2.40
Time not drawing power	13 minutes	5.00 per hour	1.0833	20%	1.30
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Total			8.5833		9.80

```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 2.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }, {
      "type": "TIME",
      "price": 5.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
```

```
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
}
```

9.3.2.17. Tariff example with reservation price and expire fee

- Reservation
  - € 4.00 reservation expiration fee (excl. VAT) (*billed when a reservation expires and is not followed by a charging session*)
  - € 2.00 per hour (excl. VAT)
  - 20% VAT
- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT

This example is very similar to Tariff example with reservation price with the difference that expired reservations cost something and that the price for reservation is different.

For a charging session that was started 22 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 6.23 (excl. VAT) or € 6.98 (incl. VAT).

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time	22 minutes	2.00 per hour	0.7333	20%	0.88

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Total			6.23		6.98

If the driver did not start a charging session and the reservation expired after the reserved time of 1 hour, the tariff would have resulted in costs of € 6.50 (excl. VAT) or € 7.80 (incl. VAT). In case a reservation is not used, the driver has to pay the full amount of reserved time as well as an additional expiration fee as compensation for not charging at all.

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Flat	1	4.00	4.00	20%	4.80
Flat	1	0.50	0.50	20%	0.60
Time	60 minutes	2.00 per hour	2.00	20%	2.40
Total			6.50		7.80

```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "FLAT",
      "price": 4.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }],
    "restrictions": {
      "reservation": "RESERVATION_EXPIRES"
    }
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 2.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "taxes": [
        {
          "name": "VAT",
```

```
          "percentage": 20
        }
      ]
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
}
```

9.3.2.18. Tariff example with reservation time and expire time

- Reservation
  - € 3.00 per hour (excl. VAT)
  - € 6.00 per hour (excl. VAT) (*billed when a reservation expires and is not followed by a charging session*)
  - 20% VAT
- Start or transaction fee
  - € 0.50 (excl. VAT)
  - 20% VAT
- Energy
  - € 0.25 per kWh (excl. VAT)
  - 10% VAT

This example is very similar to Tariff example with reservation price with the difference that expired reservations cost something and that the price for reservation is different.

For a charging session that was started 22 minutes after the reservation time, where the driver charges 20 kWh, this tariff will result in costs of € 6.60 (excl. VAT) or € 7.42 (incl. VAT).

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time	22 minutes	3.00 per hour	1.10	20%	1.32
Flat	1	0.50	0.50	20%	0.60
Energy	20 kWh	0.25 per kWh	5.00	10%	5.50
Total			6.60		7.42

If the driver did not start a charging session and the reservation expired after the reserved time of 1.5 hours, the tariff would have resulted in costs of € 9.50 (excl. VAT) or € 11.40 (incl. VAT). In case a reservation is not used, the driver has to pay the expiration fee as compensation for not charging at all.

A breakdown of this scenario is as follows:

Dimension	Quantity	Price ex VAT	Cost ex VAT	VAT	Cost inc VAT
Time	90 minutes	6.00 per hour	9.00	20%	10.80
Flat	1	0.50	0.50	20%	0.60
Total			9.50		11.40


```
{
  "currency": "EUR",
  "elements": [{
    "price_components": [{
      "type": "TIME",
      "price": 6.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }],
    "restrictions": {
      "reservation": "RESERVATION_EXPIRES"
    }
  }, {
    "price_components": [{
      "type": "TIME",
      "price": 3.00,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }],
    "restrictions": {
      "reservation": "RESERVATION"
    }
  }, {
    "price_components": [{
      "type": "FLAT",
      "price": 0.50,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 20
        }
      ]
    }, {
      "type": "ENERGY",
      "price": 0.25,
      "taxes": [
        {
          "name": "VAT",
          "percentage": 10
        }
      ]
    }]
  }],
  "last_updated": "2019-02-03T17:00:11Z"
}
```

9.4. Remote Procedure Calls on Tariff objects

The Tariffs module does not define any Remote Procedure Calls on Tariff objects.

9.5. Object type definitions

9.5.1. DayOfWeek *enum*

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

9.5.2. Price *class*

Property	Type	Card.	Description
before_taxes	decimal	1	Price/Cost excluding taxes.
taxes	TaxAmount	*	All taxes that are applicable to this price and relevant to the receiver of the Session or CDR.

9.5.3. PriceComponent *class*

A Price Component describes how a certain amount of a certain dimension being consumed translates into an amount of money owed.

Property	Type	Card.	Description
type	TariffDimensionType	1	The dimension that is being priced
price	decimal	1	Price per unit (before taxes) for this dimension.
taxes	TaxPercentage	*	Applicable taxes for this tariff dimension. If omitted, no taxes applicable. Not providing any taxes may is different from 0% VAT, which would be a value of a single tax percentage with name "VAT" and percentage 0 here.

9.5.4. ReservationRestrictionType *enum*

Value	Description
RESERVATION	Used in Tariff Elements to describe costs for a reservation.
RESERVATION_EXPIRES	Used in Tariff Elements to describe costs for a reservation that expires (i.e. driver does not start a charging session before expiry_date of the reservation).

NOTE When a Tariff has both `RESERVATION` and `RESERVATION_EXPIRES` Tariff Elements, where both Tariff Elements have a `TIME` Price Component, then the time based cost of an expired reservation will be calculated based on the `RESERVATION_EXPIRES` Tariff Element.

9.5.5. Tariff *class*

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

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

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

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

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

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

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

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

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

Besides TIME and ENERGY, there is a third possible value for the dimension enumeration, FLAT. The FLAT dimension is used to represent one-time "flat" fee components in Session pricing. FLAT is different from the other two in that it is not a quantity that is consumed during the session, but a dimension that always has the value 1 for every Session. As such there are some special rules for Price Components with the FLAT dimension:

- When, for the first time during the Charging Session, a Price Component with a FLAT dimension that is in a Tariff Element without a value for the `reservation` field in its restrictions becomes active, the price of this Price Component is added to the Session cost;
- When, for the first time during the Charging Session, a Price Component with a FLAT dimension that is in a Tariff Element with `RESERVATION_EXPIRES` as the value for the `reservation` field in its restrictions becomes active, the price of this Price Component is added to the Session cost; and

- When, for the first time during the Charging Session, a Price Component with a FLAT dimension that is in a Tariff Element with `RESERVATION` as the value for the `reservation` field in its restrictions becomes active, the price of this Price Component is added to the Session cost.

In all other cases, the price of Price Components with a FLAT dimension is ignored.

NOTE

The rule about charging only for the first FLAT fee that becomes active can have unintuitive consequences. When a Tariff contains elements for a flat price component with a restriction of `"max_energy": 1.0` and another one with a restriction of `"min_energy": 1.0`, then the flat fee from the first component will be applied for every Charging Session, even for Charging Sessions in which more than 1 kWh is charged. The reason is that at the beginning of the Session, less than 1 kWh was charged, and therefore the first component was active, and therefore it is the first price component without a `reservation` restriction to become active, and therefore it is applied. The later price component is not the first to become active and therefore not applied.

NOTE

Reviewers of this draft specification will notice that this system of dealing with flat fees is probably more complicated than it has to be. Pieter Goetschalkx of Optimile made a Proposal on Github of a more straightforward way to represent Tariffs with flat fees. This proposal requires a considerable overhaul of the description of Tariffs and will make it harder for Platforms to represent the same Tariff in both OCPI 2.2.1 and 3.0 forms. The proposal is not included in this draft but we are welcoming input about how reviewers see the trade-off between ease of migration and overall simplicity here.

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

See: Free of Charge Tariff example

Property	Type	Card.	Description
currency	CiAsciiString[3]	1	ISO-4217 code of the currency of this tariff.
tariff_alt_text	DisplayText	*	List of multi-language alternative tariff info texts.
tariff_alt_url	URL	?	URL to a web page that contains an explanation of the tariff information in human readable form.
min_price	Price	?	When this field is set, a Charging Session with this tariff will at least cost this amount. This is different from a `FLAT` fee (Start Tariff, Transaction Fee), as a `FLAT` fee is a fixed amount that has to be paid for any Charging Session. A minimum price indicates that when a Charging Session is calculated to cost less than than this amount according to the `elements` field, the cost of the Session will be equal to this amount. (Also see note below)
max_price	Price	?	When this field is set, a Charging Session with this tariff will NOT cost more than this amount. (See note below)
elements	TariffElement	+	List of Tariff Elements.
energy_mix	EnergyMix	?	Details on the energy supplied with this tariff.

Property	Type	Card.	Description
last_updated	DateTime	1	Timestamp when this Tariff was first issued.

NOTE

`min_price`: As the taxes on a Charging Session might be different for different parts of the Session, there might be situations where the minimum cost of a certain tax is reached earlier or later than the minimum price before taxes. So as a rule, they all apply: - The total cost of a Charging Session before taxes can never be lower than the `min_price` before taxes. - The total amount due for a particular tax for the Charging Session can never be lower than the amount for that tax in the `min_price`.

NOTE

`max_price`: As the taxes on a Charging Session might be different for different parts of the Session, there might be situations where the maximum cost of a certain tax is reached earlier or later than the maximum price before taxes. So as a rule, they all apply: - The total cost of a Charging Session before taxes can never be higher than the `max_price` before taxes. - The total amount due for a particular tax for the Charging Session can never be higher than the amount for that tax in the `max_price`.

NOTE

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

NOTE

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

9.5.6. TariffElement *class*

A Tariff Element is a group of Price Components that share a set of restrictions under which they apply.

That the Price Components share the same restrictions does not mean that at any time, they either all apply or all do not apply. The reason is that applicable Price Components are looked up separately for each dimension, as described under the Tariff object. Therefore it is possible that a Price Component for one dimension is found in a Tariff Element that occurs earlier in the list of Tariff Elements than for another dimension.

Property	Type	Card.	Description
price_components	PriceComponent	+	List of Price Components that each describe how a certain dimension is priced.
restrictions	TariffRestrictions	?	Restrictions that describe under which circumstances the Price Components of this Tariff Element apply.

9.5.7. TariffDimensionType *enum*

Value	Description
ENERGY	Defined in kWh.
FLAT	A flat fee charged for the Session.
TIME	Defined in hours.

NOTE

Although OCPI defines that the prices for quantities be given in the units listed in the table above in OCPI messages, Parties may convert to other units for display to certain audiences. For example, a CPO may charge a loitering fee of € 0.50 per minute for occupying a DC fast charger without charging. The CPO and eMSPs' Driver apps may show this as € 0.50 per minute even when it has to be transferred in OCPI as € 30 per hour.

9.5.8. TariffRestrictions *class*

A `TariffRestrictions` object describes if and when a Tariff Element becomes active or inactive during a Charging Session.

These restrictions are not to be interpreted as making the Tariff Element applicable or not applicable for the entire Charging Session.

When more than one restriction is set, they are to be treated as a logical AND. So a Tariff Element is active if and only if all of the properties in its `TariffRestrictions` match.

Property	Type	Card.	Description
start_time	CiAsciiString[5]	?	Start time of day in local time, the time zone is defined in the `time_zone` field of the Location, for example 13:30, valid from this time of the day. Must be in 24h format with leading zeros. Hour/Minute separator: ":" Regex: `([0-1][0-9]|2[0-3]):[0-5][0-9]`
end_time	CiAsciiString[5]	?	End time of day in local time, the time zone is defined in the `time_zone` field of the Location, for example 19:45, valid until this time of the day. Same syntax as `start_time`. If end_time < start_time then the period wraps around to the next day. To stop at end of the day use: 00:00.
min_energy	decimal	?	Minimum consumed energy in kWh, for example 20, valid from this amount of energy (inclusive) being used.
max_energy	decimal	?	Maximum consumed energy in kWh, for example 50, valid until this amount of energy (exclusive) being used.

Property	Type	Card.	Description
min_current	decimal	?	Sum of the minimum current (in Amperes) over all phases, for example 5. When the EV is charging with more than, or equal to, the defined amount of current, this TariffElement is/becomes active. If the charging current is or becomes lower, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the minimum current over the entire Charging Session. This restriction can make a TariffElement become active when the charging current is above the defined value, but the TariffElement MUST no longer be active when the charging current drops below the defined value.
max_current	decimal	?	Sum of the maximum current (in Amperes) over all phases, for example 20. When the EV is charging with less than the defined amount of current, this TariffElement becomes/is active. If the charging current is or becomes higher, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the maximum current over the entire Charging Session. This restriction can make a TariffElement become active when the charging current is below this value, but the TariffElement MUST no longer be active when the charging current raises above the defined value.
min_power	decimal	?	Minimum power in kW, for example 5. When the EV is charging with more than, or equal to, the defined amount of power, this TariffElement is/becomes active. If the charging power is or becomes lower, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the minimum power over the entire Charging Session. This restriction can make a TariffElement become active when the charging power is above this value, but the TariffElement MUST no longer be active when the charging power drops below the defined value.
max_power	decimal	?	Maximum power in kW, for example 20. When the EV is charging with less than the defined amount of power, this TariffElement becomes/is active. If the charging power is or becomes higher, this TariffElement is not or no longer valid and becomes inactive. This describes NOT the maximum power over the entire Charging Session. This restriction can make a TariffElement become active when the charging power is below this value, but the TariffElement MUST no longer be active when the charging power raises above the defined value.
min_duration	integer	?	Minimum duration in seconds the Charging Session MUST last (inclusive). When the duration of a Charging Session is longer than the defined value, this TariffElement is or becomes active. Before that moment, this TariffElement is not yet active.
max_duration	integer	?	Maximum duration in seconds the Charging Session MUST last (exclusive). When the duration of a Charging Session is shorter than the defined value, this TariffElement is or becomes active. After that moment, this TariffElement is no longer active.

Property	Type	Card.	Description
min_restrictions_duration	integer	?	Minimum duration in seconds that the other restrictions for this TariffElement must have been fulfilled. When the other restrictions of the TariffElement have been fulfilled for this many seconds, this TariffElement becomes active.
day_of_week	DayOfWeek	*	Which day(s) of the week this TariffElement is active.
reservation	ReservationRestrictionType	?	When this field is present, the TariffElement describes reservation costs. A reservation starts when the reservation is made, and ends when the driver starts charging on the reserved EVSE/Location, or when the reservation expires. A reservation can only have: FLAT and TIME TariffDimensions, where `TIME` is for the duration of the reservation.
vehicle_requesting_power	boolean	?	Restricts the applicability of the PriceComponent to situations where the vehicle is requesting power from the EVSE, or to situations where the vehicle is not requesting power. Note that the difference between `"vehicle_requesting_power": false` and something like `"max_power": 0.01` is that the former only applies when the vehicle itself indicates towards the EVSE that it will not take more energy. `"max_power": 0.01` would also apply when the EVSE is not delivering energy while the vehicle is asking for it, as can be the case due to local shortage of electric power for example.

9.5.8.1. Example: Tariff with max_power Tariff Restrictions

Example Tariff to explain the `max_power` Tariff Restriction:

- Charging fee of € 0.20 per kWh (excl. VAT) when charging with a power of less than 16 kW.
- Charging fee of € 0.35 per kWh (excl. VAT) when charging with a power between 16 and 32 kW.
- Charging fee of € 0.50 per kWh (excl. VAT) when charging with a power above 32 kW (implemented as fallback tariff without Restriction).

For a charging session where the EV charges the first kWh with a power of 6 kW, increases the power to 48 kW for the next 40 kWh and reduces it again to 4 kW after that for another 0.5 kWh (probably due to physical limitations, i.e. temperature of the battery), this tariff will result in costs of € 20.30 (excl. VAT). The costs are composed of the following components:

- 1 kWh at 6 kW: € 0.20
- 40 kWh at 48 kW: € 20.00
- 0.5 kWh at 4 kW: € 0.10

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "1",
  "currency": "EUR",
  "type": "REGULAR",
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.20,
```

```
      "vat": 20.0
    }],
    "restrictions": {
      "max_power": 16.00
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
      "price": 0.35,
      "vat": 20.0
    }],
    "restrictions": {
      "max_power": 32.00
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
      "price": 0.50,
      "vat": 20.0
    }]
  }],
  "last_updated": "2018-12-05T12:01:09Z"
}
```

9.5.8.2. Example: Tariff with max_duration Tariff Restrictions

A supermarket wants to allow their customer to charge for free. As most customers will be out of the store in 20 minutes, they allow free charging for 30 minutes. If a customer charges longer than that, they will charge them the normal price per kWh. But as they want to discourage long usage of their EVSEs, charging becomes much more expensive after 1 hour:

- First 30 minutes of charging is free.
- Charging fee of € 0.25 per kWh (excl. VAT) after 30 minutes.
- Charging fee of € 0.40 per kWh (excl. VAT) after 60 minutes.

For a charging session with a duration of 40 minutes where 5 kWh are charged during the first 30 minutes and another 1.2 kWh in the remaining 10 minutes of the session, this tariff will result in costs of € 0.30 (excl. VAT). The costs are composed of the following components:

- 5 kWh for free: € 0.00
- 1.2 kWh at 0.25/kWh: € 0.30

```
{
  "country_code": "DE",
  "party_id": "ALL",
  "id": "2",
  "currency": "EUR",
  "type": "REGULAR",
  "elements": [{
    "price_components": [{
      "type": "ENERGY",
      "price": 0.00,
      "vat": 20.0
    }],
    "restrictions": {
      "max_duration": 1800
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
```

```
      "price": 0.25,
      "vat": 20.0
    }],
    "restrictions": {
      "max_duration": 3600
    }
  }, {
    "price_components": [{
      "type": "ENERGY",
      "price": 0.40,
      "vat": 20.0
    }]
  }],
  "last_updated": "2018-12-05T13:12:44Z"
}
```

9.5.9. TaxAmount *class*

Property	Type	Card.	Description
name	UnicodeString[1..50]	1	The name of the tax. Although up to 50 characters are technically allowed here, the intention is that Parties use short names where possible, preferring e.g. "VAT" over "Value Added Tax".
amount	decimal	1	The amount of money of this tax that is due.

9.5.10. TaxPercentage *class*

Property	Type	Card.	Description
name	UnicodeString[1..50]	1	The name of the tax. Although up to 50 characters are technically allowed here, the intention is that Parties use short names where possible, preferring e.g. "VAT" over "Value Added Tax".
percentage	number	1	The applicable tax percentage.

10. Tariff Associations

This chapter describes the Tariff Associations module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the Tariff Associations module, the type of data object is the Tariff Assocation object. A Tariff Association object describes which Tariff applies at an EVSE for a certain audience from a certain time onward.

10.1. Changes from OCPI 2.2.1

- The Tariff Associations module is new in OCPI 3.0. In OCPI 2.2.1 and earlier, the functionality of the Tariff Associations module was embedded in the Tariffs module.

10.2. Replicating Tariff Associations objects

10.2.1. UC: 10.01 - Replicate Tariff Association objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains and maintains up-to-date copies of which Tariff applies for whom at the charging infrastructure offered by Party Y
2	**Description**	Using the use cases of the Party-Issued Objects chapter, Tariff Association objects are replicated from Party Y to Party X
3	**Actors**	eMSP, CPO, NAP, NSP
4	**Flow**	1. Party X subscribes to Party Y's Tariff Association objects 2. Party Y pushes all their Tariff Association objects as of subscription time to Party X 3. Party Y pushes every newly updated Tariff Association object to Party X as soon as these updates happen 4. This continues until either party cancels the subscription
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Tariff Associations module to Party X on Platform A.
6	**Postconditions**	Party X has up-to-date information on the Tariff Association objects of Party Y
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 45. UC: 10.01 Requirements

ID	Precondition	Requirement
R.10.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "tariffassociations" as the ModuleID value.

ID	Precondition	Requirement
R.10.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.10.01.01
R.10.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.10.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the TariffAssociation object type.
R.10.01.04		Tariff Association objects are immutable. That is, Platform B SHALL NOT send an update with a changed version number for a previously received object using use case Send a Full Update of a Party-Issued Object to a Subscribed Platform in the context of a subscription to the Tariff Associations module.
R.10.01.05	Platform A receives an update with a changed version number for a previously received object using use case Send a Full Update of a Party-Issued Object to a Subscribed Platform	Platform A SHALL respond with an OcpiResponse with no value for the `payload` field and with the `status_code` field set to 6003.

NOTE

OCPI 3.0 does not impose limits on how frequently a Tariff Associations sender can send Tariff Associations, nor on which Tariff Associations can be sent when. Nevertheless, Tariff Associations senders have to make sure that the Tariff Association receivers can reasonably know which Tariff will apply when a charging session is started at a certain time. Tariff Associations senders will typically apply constraint in regards to the following aspects of their sending timing:

- The frequency of updates from CPO to eMSP
- The frequency of price changes from a Driver's perspective
- Notice time before the Tariff takes effect from the CPO to the eMSP
- Notice time before the Tariff takes effect to the Driver

As of the time of writing, white papers with practical advice on how to manage Tariff communication are available from the EV Roaming Foundation at https://evroaming.org/white-papers/.

10.3. Remote Procedure Calls on Tariff Association objects

The Tariff Associations module does not define any Remote Procedure Calls on Tariff Association objects.

10.4. Other Tariff Associations use cases

10.4.1. UC: 10.02 - Cancel a Tariff Association

1	**Objective(s)**	**1.** Party Y on Platform B makes sure that a Tariff Association that they previously sent to Party X on Platform A will not be used anymore by Party X to inform Drivers or to check CDRs' and Sessions' costs.
2	**Description**	Sometimes a Party issues a Tariff Association in some sort of error, and they want to cancel or undo the issuing in some way. With the Tariff Association system, doing so is not entirely straightforward, so this use case describes how to do it.
3	**Actors**	CPO, eMSP
4	**Flow**	1. Party Y on Platform B pushes one or more Tariff Associations to Party X on Platform A that nullify the effect of a Tariff Association pushed earlier to Party X by Party Y.
5	**Preconditions**	Party X hosted on Platform A is subscribed to Tariffs of Party Y hosted on Platform B. Party X is subscribed to Tariff Associations of Party Y. Party X received a Tariff Association from Party Y. Party Y now wishes to cancel that Tariff Association.
6	**Postconditions**	Party X no longer uses the Tariff Association to inform Drivers or to check Session costs.
7	**Error handling**	None specified.
8	**Remark(s)**	

Table 46. UC: 10.02 Requirements

ID	Precondition	Requirement
R.10.02.01		Party Y sends one or more Tariff Associations according to Replicate Tariff Association objects.
R.10.02.02		The union of the values of the `connectors` fields of the Tariff Association objects mentioned in R.10.02.01 is the same as the value of the `connectors` field of the Tariff Association that Party Y is canceling.
R.10.02.03		Each of the Tariff Association objects mentioned in R.10.02.01 has a value for the `start_date_time` field that is the same as the value of `start_date_time` field in the Tariff Association that Party Y is canceling.
R.10.02.04		Each of the Tariff Association objects mentioned in R.10.02.01 has a value for the `audience` field that is the same as the value of the `audience` field in the Tariff Association that Party Y is canceling.

10.5. Object type definitions

10.5.1. ConnectorReference *class*

A *ConnectorReference* uniquely identifies a Connector that a Tariff Association applies a Tariff for.

A Tariff Association can only refer to connectors issued by the same party that issued the Tariff Association. The party

ID of the connectors is therefore not explicitly given in the ConnectorReference objects.

Property	Type	Card.	Description
evse_uid	CiAsciiString[1..36]	1	The UID of the EVSE in which the connector is that this ConnectorReference refers to
connector_id	CiAsciiString[1..36]	1	The ID of the connector that this ConnectorReference refers to.

10.5.2. TariffAssociation *class*

Property	Type	Card.	Description
start_date_time	DateTime	1	The timestamp at which this Tariff Association comes into effect (inclusive)
tariff_id	CiAsciiString[1..36]	1	The ID of the Tariff that is applied by this Tariff Association.
connectors	ConnectorReference	+	The identifiers of the connectors that this Tariff Association applies a Tariff to. The receiver SHALL NOT send an error response when it receives a Tariff Association object referencing connectors that it did not yet receive via the Locations module. It SHOULD instead store these dangling references as such and attempt to resolve them once it is looking up a Tariff for a Session.
audience	TariffAudience	1	The audience (MSP contract holders, ad-hoc paying drivers, ...) that the Tariff Association applies a Tariff for.
last_updated	DateTime	1	The timestamp when this Tariff Association was last updated or created by the Party issuing it.

10.5.3. TariffAudience *enum*

Value	Description
AD_HOC_PAYMENT	Used to describe that a Tariff Association applies when ad-hoc payment is used at the Charging Station (for example: Debit or Credit card payment terminal).
PROFILE_CHEAP	Used to describe that a Tariff Association applies when Charging Preference `CHEAP` is set for the session.
PROFILE_FAST	Used to describe that a Tariff Association applies when Charging Preference: `FAST` is set for the session.
PROFILE_GREEN	Used to describe that a Tariff Association applies when Charging Preference: `GREEN` is set for the session.
REGULAR	Used to describe that a Tariff Association applies when using an MSP Charging Token, without any Charging Preference, or when Charging Preference: `REGULAR` is set for the session.

11. Tokens

This chapter describes the Tokens module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the Tokens module, the type of data object is the Token. The name Token is short for Charge Token. What a Charge Token is is defined in the Terminology section of the Functional Use Cases document. How a Charge Token is represented in OCPI messages is specified below.

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

Besides this mechanism of authorizing Charge Sessions based on cached Token information, the Tokens module also offers a use case for *real-time authorization*, which allows CPOs to request authorization for a Charge Session start from eMSPs right at the moment that a Driver tries to start charging. OCPI allows for eMSPs to not share their Charge Tokens with CPOs and instead rely completely on real-time authorization.

eMSPs control which form of authorization, real-time or cache-based, is used by the whitelist field of the Token objects that they issue. There are, broadly speaking, three approaches for eMSPs:

- Not issue any Tokens via OCPI, that is, not share any Token objects with CPOs. This means that all Charge Sessions started with the eMSP's Tokens by a Driver interacting directly with a Charging Station will have to be authorized through real-time authorization. Also CPOs will have to broadcast authorization requests to all eMSPs because they can't tell to which eMSP the Token belongs when they receive the authorization request from the Charging Station. This option minimizes the upfront sharing of personal information but is the least performant and resilient. Also it leads to unnecessary sharing of personal and competition sensitive information with the broadcast authorization requests. For EMAID tokens authenticated with ISO 15118 contract certificates, no broadcasting of authorization reuqests is necessary with this approach, and this approach becomes very sensible.
- Issue tokens via OCPI and set the `whitelist` field to `NEVER`. This means that all Charge Sessions started with the eMSP's Tokens by a Driver interacting directly with a Charging Station will have to be authorized through real-time authorization. But unlike when the eMSP does not issue Tokens at all, the CPO will be able to tell immediately which eMSP they have to contact for authorization.
- Issue tokens via OCPI and set the `whitelist` field to `ALWAYS`. This means that Charge Sessions with the eMSP's Tokens by a Driver interacting directly with a Charging Station can be authorized by the CPO without contacting the eMSP at authorization time. This option is the most performant and resilient but requires the eMSP to share some customer data with CPOs and makes the eMSP dependent on the CPO's cooperation to invalidate Tokens on that CPO's network.

If an eMSP goes for the third option, and never issues Tokens with `whitelist` set to anything else than `ALWAYS`, their OCPI platform does not have to implement real-time authorization.

11.1. Changes from OCPI 2.2.1

- Removed the country code and party ID, as from all Party Issued Objects, because these are now unambiguously transferred using the Party Issued Object replication mechanism
- OCPI 3.0 no longer allows an MSP to give real-time authorization for a certain set of EVSEs only, as existed in OCPI 2.2.1 with the `location` field of the `AuthorizationInfo` object. The idea is that with OCPI 3.0, CPOs should send

their authorize requests as specific as possible with the CPO's knowledge, so there will be no way for a CPO to enforce additional constraints from the eMSP.

- In a real-time authorization response, the eMSP can now tell the CPO what Tariff they will charge the Driver, so the CPO can display this on the Charging Station or pass it on to the vehicle.
- The `token_id` field in the AuthorizeRequest object was renamed to `token_uid`.
- Timestamps fields were added to real-time authorization requests and responses to make real-time authorizations more traceable and more enforceable.
- Fields were added to let the eMSP set limits on time and energy consumption in the real-time authorization response.

11.2. Replicating Token objects

11.2.1. UC: 11.01 - Replicate Token objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains and maintains an up-to-date copy of the Token objects that are issued by Party Y on a Platform B
2	**Description**	Using the use cases of the Party-Issued Objects chapter, Token objects are replicated from Party Y to Party X
3	**Actors**	eMSP, CPO, Hub
4	**Flow**	1. Party X subscribes to Party Y's Token objects 2. Party Y pushes all their Token objects as of subscription time to Party X 3. Party Y pushes every newly updated Token object to Party X as soon as these updates happen 4. This continues until either party cancels the subscription
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Tokens to Party X on Platform A.
6	**Postconditions**	Party X has up-to-date information on the Charge Tokens issued by Party Y
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 47. UC: 11.01 Requirements

ID	Precondition	Requirement
R.11.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "tokens" as the ModuleID value.

ID	Precondition	Requirement
R.11.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.11.01.01
R.11.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.11.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the Token object type.
R.11.01.04	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.11.01.01	Platform B SHALL set the `id` field of the PartyIssuedObjectUpdate object in the request body to a string that uniquely identifies the token among all Tokens issued by Party Y. Party Y MAY use the value of the Token's `uid` field as the value for the `id` field in PartyIssuedObjectUpdate if Party Y can guarantee that the `uid` values of their Tokens are unique among all Tokens they issued. Party Y MAY also use generated unique IDs, in which case Party Y can issue multiple Tokens with the same value for the `uid` field.

11.2.2. Example token objects

11.2.2.1. Example APP_USER token

```
{
  "uid": "bdf21bce-fc97-11e8-8eb2-f2801f1b9fd1",
  "type": "APP_USER",
  "contract_id": "DE8ACC12E46L89",
  "issuer": "TheNewMotion",
  "valid": true,
  "whitelist": "ALLOWED",
  "last_updated": "2018-12-10T17:16:15Z"
}
```

11.2.2.2. Example RFID token

```
{
  "uid": "12345678905880",
  "type": "RFID",
  "contract_id": "DE8ACC12E46L89",
  "visual_number": "DF000-2001-8999-1",
  "issuer": "TheNewMotion",
  "group_id": "DF000-2001-8999",
  "valid": true,
  "whitelist": "ALLOWED",
  "language": "it",
  "default_profile_type": "GREEN",
  "energy_contract": {
    "supplier_name": "Greenpeace Energy eG",
    "contract_id": "0123456789"
```

```
  },
  "last_updated": "2018-12-10T17:25:10Z"
}
```

11.2.2.3. Example EMAID token

```
{
  "uid": "DE8ACC12E46L89",
  "type": "EMAID",
  "contract_id": "DE8ACC12E46L89",
  "issuer": "TheNewMotion",
  "group_id": "DF000-2001-8999",
  "valid": true,
  "whitelist": "NEVER",
  "language": "it",
  "default_profile_type": "GREEN",
  "energy_contract": {
    "supplier_name": "Greenpeace Energy eG",
    "contract_id": "0123456789"
  },
  "last_updated": "2023-12-15T10:15:10Z"
}
```

11.3. Remote Procedure Calls on Token Objects

11.3.1. UC: 11.02 - Ask for real-time charge authorization

1	**Objective(s)**	**1.** Party X knows that Party Y authorizes a Charge Session with one of Party Y's tokens on one of the Locations issued by Party X
2	**Description**	A remote procedure call is made by Party X to Party Y. In the request Party X sends the specifics of the Charge Session that they wish to start. Party Y then responds, informing Party X of whether they authorize the Session or not.
3	**Actors**	eMSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the Charge Token that the Driver is offering, the Location ID of the Location on which Party X wants to start a Session, a timestamp of when the CPO first learned of the authorization request, and optionally a list of Connector IDs on which the Session may be started if authorized. 2. Party Y decides if they authorize a Session with the given details. 3. Party Y sends a response to Party X informing Party X of Party Y's decision.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Tokens to Party X on Platform A.
6	**Postconditions**	Party X knows Party Y authorizes a session with the given details.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.

8	**Remark(s)**	This use case is explicitly not meant for real-time tariff negotiations. While one could imagine adding a Tariff field to the authorization request and a "too expensive" status code to the response, such proposals were discussed and decided against during OCPI 3.0 development. In the OCPI Development Working Group there was a consensus at the time that the industry is currently not ready for automated real-time pricing negotiation between CPO and eMSP.

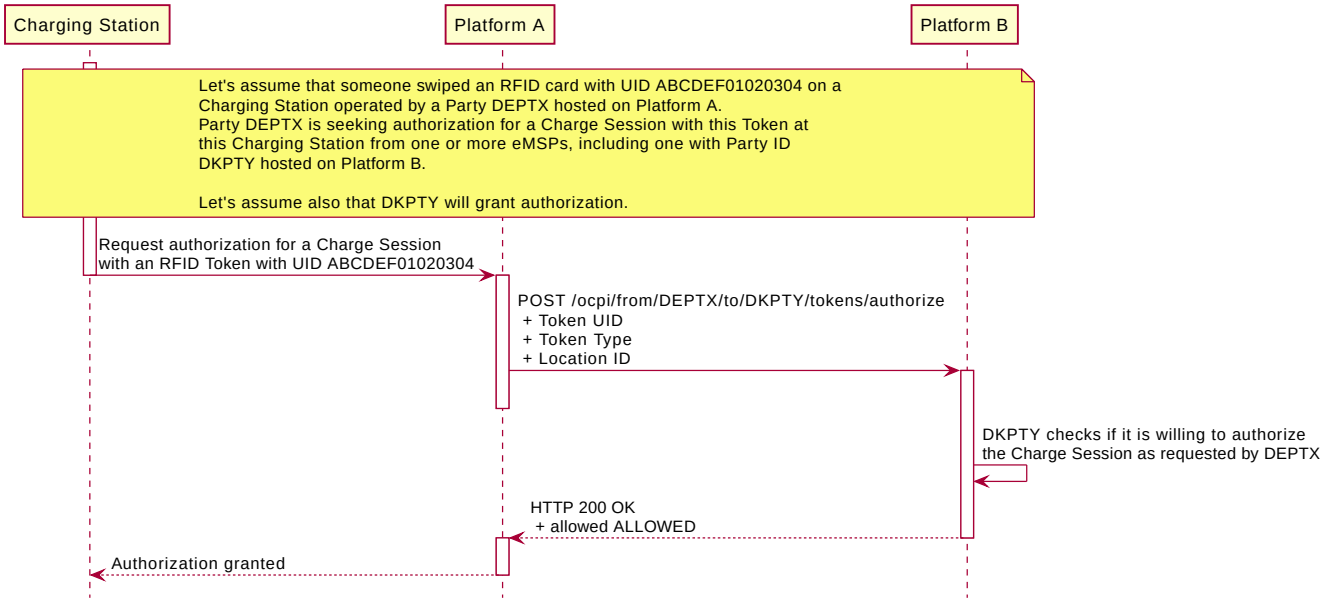


Figure 47. Sequence Diagram: Ask for real-time charge authorization

Table 48. UC: 11.02 Requirements

ID	Precondition	Requirement
R.11.02.01		Platform A SHALL make the request to reserve an EVSE following Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.11.02.02		Platform A SHALL use "authorize" as the operation name for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.11.02.03		Platform A SHALL use POST as the HTTP request verb when making its request.
R.11.02.04		Platform A SHALL use a AuthorizeRequest in the payload field of the request body for Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.
R.11.02.05		Platform B SHALL include a AuthorizeResponse in the payload field of the OcpiResponse object in the response body according to Make a Remote Procedure Call on behalf of a Party to another Party on another Platform.

ID	Precondition	Requirement
R.11.02.06		Party Y SHOULD NOT validate that charging is possible based on information shared by Party X via the Location module, like opening hours or EVSE status, etcetera. Such information might not be up to date and it is not the eMSP's responsibility to check if a Session is possible on the CPO side.
R.11.02.07	Party X wants to start a transaction with a Token issued by Party Y on a Location that it did not issue to Party Y	Party X MAY send an AuthorizeRequest containing a Location ID of a Location that was never shared with Party Y
R.11.02.08	Party Y does not recognize the pair of token UID and token type given in the request from Platform A	Platform B SHALL set the `status_code` field in the response to 7001 following the Make a Remote Procedure Call on behalf of a Party to another Party on another Platform use case.

11.4. Object type definitions

11.4.1. AllowedType *enum*

Value	Description
ALLOWED	This Token is allowed to charge (at the given 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).
REQUEST_TOO_BROAD	This Token is not allowed to charge at the whole Location or all of the Connectors in the request, but may be authorized to charge on a subset of the Connectors that authorization was requested for.
NOT_ALLOWED	The Token not allowed to charge at the Location or at any of the Connectors that authorization was requested for.

11.4.2. AuthorizeRequest *class*

Property	Type	Card.	Description
token_uid	CiAsciiString[1..255]	1	Token UID of the Token for which authorization is requested
type	TokenType	?	Type of the Token that authorization is requested for. By default this is RFID.

Property	Type	Card.	Description
presentation_timestamp	DateTime	1	The time at which the CPO first learned of the authorization request with this token. If possible, this could be the timestamp at which a token was presented to the Charging Station on site, like when a physical RFID token was presented to an RFID reader, or or when a contract certificate was read from a vehicle. If the Charging Station does not provide the CPO with such timestamps, the CPO can use the timestamp of when they received the authorization request from the Charging Station.
location_id	CiAsciiString[1..255]	1	The ID of the Location on which the CPO is starting the Charge Session that it is seeking authorization for.
evse_uids	CiAsciiString[1..36]	*	A list of UIDs of EVSEs. If this is set, it means that the CPO is seeking authorization to start a Session on one of these EVSEs. The idea behind this field is that if a CPO receives an OCPP 1.6 Authorize request from a Charging Station, it knows that the session will happen on one of the EVSEs of that Charging Station but it will not be sure which one of those EVSEs it will be. This field allows the CPO to share its knowledge of which EVSEs the authorization request applies to with the eMSP. OCPI is using a list of EVSE UIDs here instead of a ChargingStation ID because OCPP 1.6 Charge Points are not necessarily mapped one-to-one to OCPI Charging Stations.

NOTE although the `presentation_timestamp` field is included for traceability, CPOs are not expected to use the real-time authorization to obtain retroactive authorization for Sessions that have already started. If CPOs do so anyway, eMSPs are expected to reject such authorization requests.

11.4.3. AuthorizeResponse *class*

Property	Type	Card.	Description
allowed	AllowedType	1	Status of the Token, and whether charging is allowed at the Location given in the corresponding AuthorizationRequest, possibly restricted to the EVSEs given in the AuthorizationRequest.
token	Token	1	The complete Token object for which this authorization was requested.
authorization_reference	CiAsciiString[1..36]	?	Reference to the authorization given by the eMSP. When given, this reference will be provided in the relevant Session and CDR.
authorization_timestamp	DateTime	1	The time from which the Token is authorized to charge at the Location given in the request. The eMSP MAY dispute the resulting CDR if the CPO uses the authorization reference for a Session that started before this timestamp.

Property	Type	Card.	Description
authorized_until	DateTime	1	The time until which the Token is authorized to charge at the Location given in the request. The eMSP MAY dispute the resulting CDR if the CPO uses the authorization reference for a Session that started at or after this timestamp.
max_energy	number	?	If given, this is the maximum amount of energy that the eMSP is authorizing the CPO to let the Driver charge. If the CPO starts a Charging Session for this authorization, and sends the eMSP a CDR for that Session with more energy consumed than the amount of kWh given in this field, then the eMSP MAY dispute the CDR. This field is meant to facilitate pre-paid eMSP models. It is typically left unfilled in other scenarios.
max_time	number	?	If given, this is the maximum amount of time that the eMSP is authorizing the CPO to let the Driver charge for. If the CPO starts a Charging Session for this authorization, and sends the eMSP a CDR for that Session than lasted longer than the amount of hours given in this field, then the eMSP MAY dispute the CDR. This field is meant to facilitate pre-paid eMSP models. It is typically left unfilled in other scenarios.
info	DisplayText	?	Optional display text, additional information to the EV driver.
display_tariff	Tariff	?	The Tariff that will be charged by the eMSP to the Driver, to be displayed on the Charging Station. This added because regulations in the US State of California require that a Driver see on the Charging Station what they will be paying when they start a Charging Session. This field can also be used in combination with IEC 15118 to provide pricing information from the Charging Station to the vehicle. Where these two use cases do not apply, this field may be left empty.

11.4.4. EnergyContract *class*

Information about a energy contract that belongs to a Token so a driver could use his/her own energy contract when charging at a Charging Station.

Property	Type	Card.	Description
supplier_name	UnicodeString[1..64]	1	Name of the energy supplier for this token.
contract_id	UnicodeString[1..64]	?	Contract ID at the energy supplier, that belongs to the owner of this token.

11.4.5. Token *class*

Property	Type	Card.	Description
uid	CiAsciiString[1..36]	1	Unique ID by which this Token, combined with the Token type, can be identified. In the case of RFID tokens, this is the UID according to ISO/IEC 14443, which is the field used by CPO system (RFID reader on the Charge Point) to identify this token. If this is a APP_USER or AD_HOC_USER Token, it will be a unique ID generated by the eMSP. If this is an EMAID Token, it will be the contract ID. This means that for Tokens with type EMAID, the fields `uid` and `contract_id` will hold the same value.
type	TokenType	1	Type of the token
contract_id	CiAsciiString[1..36]	1	Uniquely identifies the EV Driver contract within the eMSP's platform (and suboperator platforms). Recommended to follow the specification for eMA ID from "eMI3 standard version V1.0" (http://emi3group.com/documents-links/) "Part 2: business objects."
visual_number	UnicodeString[1..64]	?	Visual readable number/identification as printed on the Token (RFID card), might be equal to the contract_id.
issuer	string[1..64]	1	Issuing company, most of the times the name of the company printed on the token (RFID card), not necessarily the eMSP.
group_id	CiAsciiString[1..36]	?	This ID groups a couple of tokens. This can be used to make two or more tokens work as one, so that a session can be started with one token and stopped with another, handy when a card and key-fob are given to the EV-driver. Beware that OCPP 1.5/1.6 only support group_ids (it is called parentId in OCPP 1.5/1.6) with a maximum length of 20.
valid_from	datetime	1	A point in time from which the Token is valid, inclusive.
valid_until	datetime	?	A point in time when the validity of the token ends.
whitelist	WhitelistType	1	Indicates what type of white-listing is allowed.
language	CiAsciiString[2]	?	Language Code ISO 639-1. This optional field indicates the Token owner's preferred interface language. If the language is not provided or not supported then the CPO is free to choose its own language.
default_profile_type	ProfileType	?	The default Charging Preference. When this is provided, and a charging session is started on an Charging Station that support Preference base Smart Charging and support this ProfileType, the Charging Station can start using this ProfileType, without this having to be set via: Set Charging Preferences.

Property	Type	Card.	Description
energy_contract	EnergyContract	?	When the Charging Station supports using your own energy supplier/contract at a Charging Station, information about the energy supplier/contract is needed so the CPO knows which energy supplier to use. NOTE: In a lot of countries it is currently not allowed/possible to use a drivers own energy supplier/contract at a Charging Station.
last_updated	DateTime	1	Timestamp when this Token was last updated (or created).

The combination of *uid* and *type* should be unique for every token within the eMSP's system.

NOTE OCPP supports group_id (or ParentID as it is called in OCPP 1.5/1.6) OCPP 1.5/1.6 only support group ID's with maximum length of string(20), case insensitive. As long as EV-driver can be expected to charge at an OCPP 1.5/1.6 Charging Station, it is adviced to not used a group_id longer then 20.

11.4.6. TokenType *OpenEnum*

Value	Description
AD_HOC_USER	One time use Token ID generated by a server (or App.) The eMSP uses this to bind a Session to a customer, probably an app user.
APP_USER	Token ID generated by a server (or App.) to identify a user of an App. The same user uses the same Token for every Session.
EMAID	An EMAID. EMAIDs are used as Tokens when the Charging Station and the vehicle are using IEC 15118 for communication.
RFID	RFID Token, typically using the ISO/IEC 14443 standard.

NOTE The eMSP is RECOMMENDED to push Tokens with type `AD_HOC_USER` or `APP_USER` with `whitelist` set to `NEVER`. Whitelists are very useful for RFID type Tokens, but the `AD_HOC_USER`/`APP_USER` Tokens are used to start Sessions at the initiative of the eMSP, so the CPO whitelisting them has no advantages.

NOTE The eMSP is RECOMMENDED to not push Tokens with type `EMAID` at all. Replicating these as Party Issued Objects is not necessary because the CPO already learns which Party issued the Token from the Charging Station. The CPO can then contact this Party for real-time authorization using Use Case *Ask for real-time authorization*.

NOTE The management of contract certificates used with IEC 15118 to authenticate the vehicle are left ouside of OCPI 3.0 for now. There are other existing standards for exchanging and validating certificates that Parties can use to authenticate contract certificates.

11.4.7. WhitelistType *enum*

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

The validity of a Token has no influence on this. If a Token is not valid at the time of the authorization check according to the `valid_from` and `valid_until` fields, when the `whitelist` field requires real-time authorization, the CPO SHALL do

a real-time authorization to check if the Token is still not valid.

Value	Description
ALWAYS	Token always has to be whitelisted, realtime authorization is not possible or not allowed. CPO shall always decide on authorization of Sessions for this Token without contacting the eMSP.
ALLOWED	It is allowed to whitelist the token, realtime authorization is also allowed. The CPO may choose which version of authorization to use.
ALLOWED_OFFLINE	In normal situations realtime authorization shall be used. But when the CPO cannot get a response from the eMSP (communication between CPO and eMSP is offline), the CPO SHALL decide on authorization of a Session with this Token based on the CPO's cached Token information.
NEVER	Whitelisting is forbidden, only realtime authorization is allowed. CPO shall always send a realtime authorization for any use of this Token to the eMSP.

12. Invoice Reconciliation

This chapter describes the Invoice Reconciliation module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one Party to another. In the Invoice Reconciliation module, the type of data object is the Invoice Reconciliation Record (IRR). An IRR specifies which CDRs are listed on which invoice.

12.1. Changes from OCPI 2.2.1

The Invoice Reconciliation module provides functionality that is completely new to OCPI in version 3.0.

12.2. High-level description

This section is not normative.

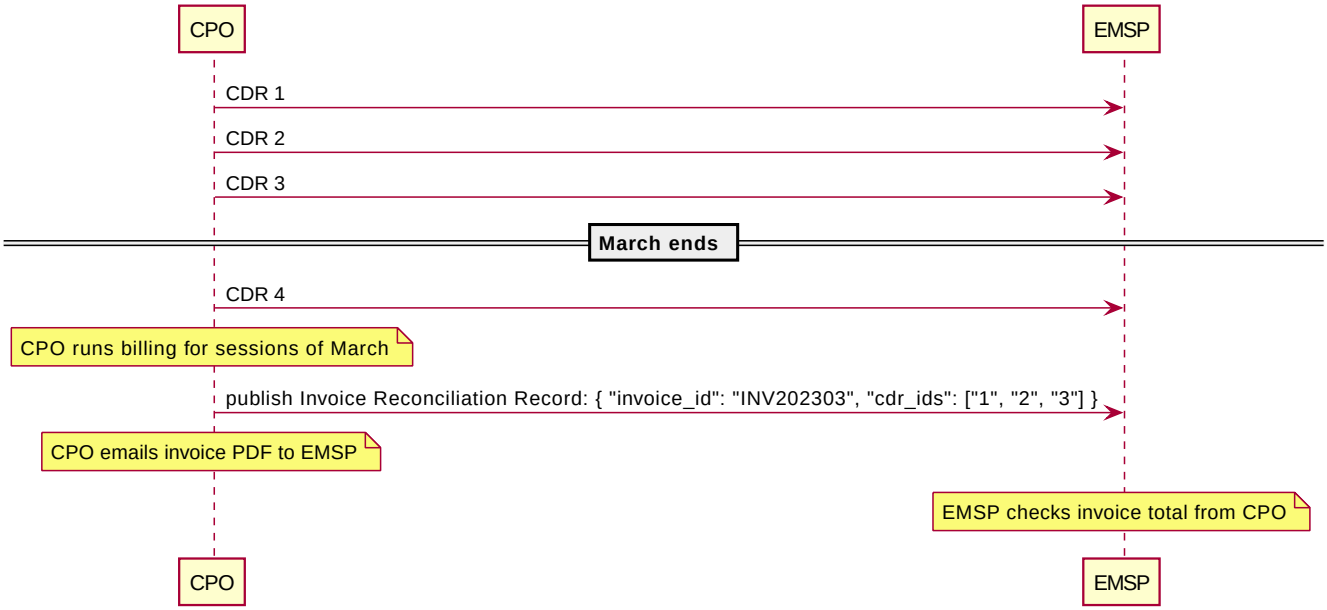
The Invoice Reconciliation serves to enable Parties that receive invoices for Charging Sessions to check the amounts of these invoices against the CDR data that they transferred via OCPI.

It is deliberately flexible so that Parties that invoice among each other, typically CPOs and EMPS, are still free to decide:

- When they invoice (for example, with a monthly billing cycle or with a new invoice for every Charging Session)
- How they transfer the actual invoice documents (postal mail, e-mail, or purpose-made IT solutions)
- How they pay the invoices

The workflow for invoice reconciliation will work somewhat differently depending on whether the invoicing Parties use *direct billing* or *reverse billing*. Direct billing is the situation in which the Party who delivered the service, that is typically the CPO, sends invoices to the Party who consumed the service, that is typically the MSP. Reverse billing is the situation in which the Party who consumed the service, that is typically the eMSP, sends invoices for credit amounts to the Party who delivered the service, that is typically the CPO.

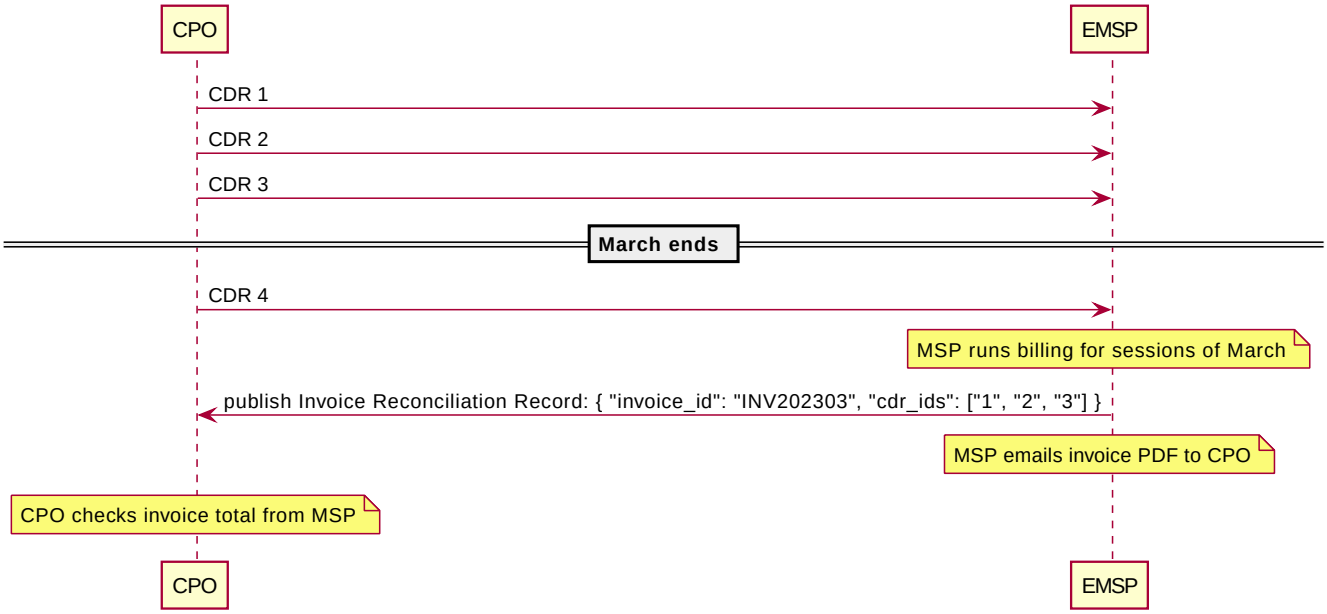
With direct billing the flow for Invoice Reconciliation looks like this:



What we see here is a process with the following steps:

- The CPO conducts Charging Sessions and issues CDRs for them to the eMSP
- At some point, at the CPO's discretion, the CPO runs billing and produces an invoice for the Charging Sessions. In the diagram, as an example, we assume that the CPO runs a monthly billing cycle and the end of the month of March is the trigger for them to produce an invoice.
- Once the CPO completes the invoice, it publishes an Invoice Reconciliation Record of it to the MSP
- Note that in the example, a CDR 4 is delivered between the moment March ends and the Invoice Reconciliation Record for the March invoice is published. It is published before the IRR is published, but is nevertheless not mentioned in the IRR and not invoiced by the invoice that the IRR is about. This CDR 4 in the xample serves to illustrate that the set of invoices referenced by an Invoice Reconciliation Record is not determined by timing, but by the list of invoice IDs in the IRR.
- The CPO also delivers the invoice to the eMSP. How precisely this happens is not relevant, as long as the eMSP's staff are able to access the invoice document.
- When the eMSP has both the IRR and the invoice, they can easily reconcile by computing the total amount for the IRR's CDRs in their own systems, and comparing this computed amount to the amount listed in the invoice document.

When we change the flow so that reverse billing is used, and all other aspects remain the same, we get this flow:



We see here that the billing cycle is started at the eMSP instead of the CPO, and from that point on, the communication pattern between the two Parties is exactly reversed compared to the direct billing scenario. Although accordingly the roles of Invoice Reconciliation Record sender and receiver are reversed among the CPO and the eMSP, the very same technical OCPI interface can be used for both scenarios. The following normative parts of the specification of this module make no distinction between direct and reverse billing. Nevertheless, both approaches are possible and the Invoice Reconciliation module can be used with both.

12.3. Replicating Invoice Reconciliation Record objects

12.3.1. UC: 12.01 - Replicate Invoice Reconciliation objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains and maintains up-to-date information of which CDRs have been invoiced with which invoices that Party Y sent to Party X
2	**Description**	Using the use cases of the Party-Issued Objects chapter, Invoice Reconciliation Records objects are replicated from Party Y to Party X
3	**Actors**	eMSP, CPO, NAP, NSP
4	**Flow**	1. Party X subscribes to Party Y's Invoice Reconciliation Records objects 2. Party Y pushes all their Invoice Reconciliation Record Objects for Party X as of subscription time to Party X 3. Party Y pushes every newly created Invoice Reconciliation Record object to Party X as soon as it sent an invoice to Party X and created the Invoice Reconciliation Record for it. 4. This continues until either party cancels the subscription
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Invoice Reconciliation module to Party X on Platform A.

6	**Postconditions**	Party X has up-to-date information on the Invoice Reconciliation Record objects that Party Y made for Party X
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 49. UC: 12.01 Requirements

ID	Precondition	Requirement
R.12.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "irrs" as the ModuleID value.
R.12.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.12.01.01
R.12.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.12.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the InvoiceReconciliationRecord object type.

12.4. Remote Procedure Calls on Invoice Reconciliation Record objects

The Invoice Reconciliation module does not define any Remote Procedure Calls on Invoice Reconciliation Record objects.

12.5. Object type definitions

12.5.1. InvoiceReconciliationRecord *class*

Property	Type	Card.	Description
invoice_id	CiAsciiString[1..255]	1	An identifier identifying an invoice sent by the party issuing this IRR. The precise format is up to the sender and receiver parties to decide. The intended workflow will work as long as the receiver can use the identifier to find the invoice that this IRR lists the CDRs for. For example, these invoice identifiers might be opaque document IDs but could also identify the invoice out of a monthly billing cycle with a string like `2023-06`.

Property	Type	Card.	Description
cdr_ids	CiAsciiString[1..255]	+	The unique CDR identifiers of the CDRs that are invoiced by the invoice identified by the value of the `invoice_id` field.
last_updated	DateTime	1	Timestamp at which this Invoice Reconciliation Record was issued

13. Power Regulation

This chapter describes the Power Regulation module.

An OCPI 3.0 module is a set of Functional Use Cases organised around a certain type of data object being replicated from one party to another. In the Power Regulation module, the type of replicated data object is the MeterSample. A MeterSample is a record of the energy, power, current, voltage and/or other quantities related to a Charging Session at a certain time.

With the Power Regulation module, parties (SCSPs but also eMSPs) can send Charging Profiles for a certain Charging Session or grouping of EVSEs to a CPO. These Charging Profiles instruct the CPO what the the rate of energy transfer should be at which time during the affected Charging Session or Charging Sessions. After sending the Charging Profiles, the SCSP or eMSP can use Party Issued Object replication to receive the Meter Samples related to the Charging Sessions that they sent Charging Profiles for. This process of influencing and monitoring the rate of energy transfer in a Charging Session is widely known as "Smart Charging".

The following sequence diagram illustrates the general flow of Smart Charging or Power Regulation in OCPI 3.0.

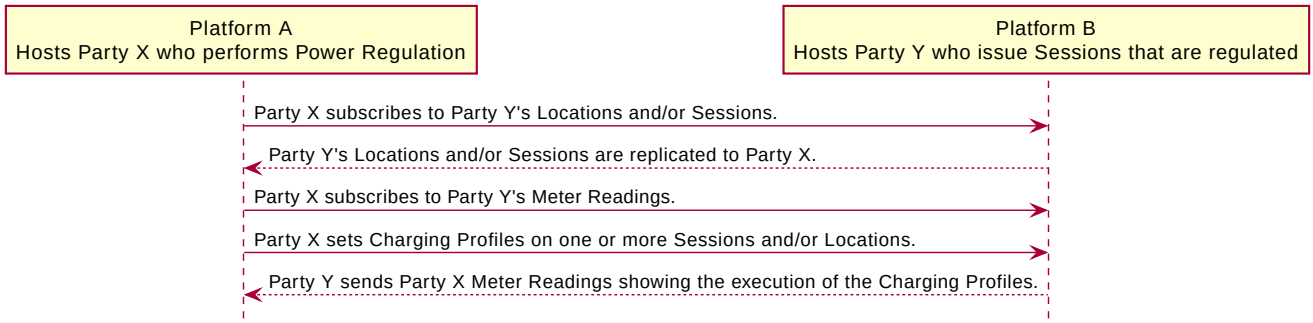


Figure 48. Sequence Diagram: General flow of Power Regulation

TODO how about V2G / bidirectional charging? Profiles only have a "maximum limit" as of now

It is also possible to request the 'ActiveChargingProfile' from a Location/EVSE where a Charging Session is ongoing.

The ActiveChargingProfile is the Charging Profile as calculated by the EVSE. It is the result of the calculation of all smart charging inputs present in the EVSE, also Local Limits might be taken into account.

The Charging Profile concept in OCPI is similar to the concept of Charging Profiles in OCPP, but exposes this functionality to third parties. These objects and the accompanying interfaces provide certain abstractions that make them more suitable for energy parties to signal their intent. The data structures are based on those in OCPP versions 1.6 and 2.0.1 to make conversion of messages between OCPI and OCPP easy.

NOTE

Charging Profiles set via this module are no guarantee that the EV will charge with the exact given limit, for it is a limit, not a target. A lot of factors influence the charging power. The EV might not take the amount of energy that the EVSE is willing to provide to it, or the battery might be too warm or almost full. A single phase cable might be used on a three phase Charging Station. There can be local energy limits (load balancing between EVSEs on a relative small energy connection to a group of EVSEs) that might limit the energy offered by the EVSE to the EV even further.

A Charging Profile can be set by the owner of a Token on Sessions that belong to that token. If another party sends a ChargingProfile and the CPO has no contract that allows that party to set profiles on sessions, the CPO is allowed to reject such profiles.

This module can be used by the eMSP, but can also be used by another party that provide "Smart Charging Services" (Smart Charging Service Provider (SCSP) / Aggregator / Energy Service Broker etc.) These SCSPs then depend on the CPO sending Locations and/or Sessions information to them. They need to know which Locations are available or which Sessions are ongoing to be able to influence them.

If an SCSP uses this module, read eMSP as SCSP.

Most Charging Stations are hooked up to the internet via a relatively slow wireless connection. To prevent long blocking calls, the calls to the CPO in the Power Regulation module are designed to work asynchronously using the Make a Remote Procedure Call Allowing Asynchronous Responses use case.

The CPO can limit the amount of request that can be done on the Power Regulation interface. This allows the CPO to prevent creating a too high load or data usages. To do this the CPO can reject a request on the Charging Profile Receiver interface be responding with: `TOO_OFTEN`.

NOTE	OCPI provides the means for SCSPs to do certain things. Parties using OCPI still have to adhere to local privacy laws, have to have agreed on contracts etc. Local laws might oblige explicit consent from the driver etc.

Module dependency: Sessions module, Locations module.

13.1. A note for the reviewers

This module has been substantially overhauled since OCPI 2.2.1. It was renamed from Charging Profiles to Power Regulation in the process.

The new name reflects a new perspective on Smart Charging compared to earlier versions of OCPI and also OCPP. Unlike these standards, OCPI 3.0 sees Smart Charging as a process of bidirectional communication. It is not just about one Party sending Charging Profiles to another Party; it is equally well about the other Party sending feedback about the actual realized charging power. Without such feedback, the Party sending the profiles would be flying blind.

This module now includes its own data format for sending data about the power consumption ongoing Charging Sessions and of EVSEs: the MeterSample object. We have chosen to separate this from the Sessions module because the typical eMSP's use case is different from the typical SCSP's, and these Parties typically have different information needs. SCSPs want to know things about the power that would be uninteresting to a Driver and their eMSP, like how many amps are flowing on which phase or how much power a whole Location is drawing from the grid. eMSPs on the other hand are interested in personally identifying information (PII) that allows them to relate the Charging Session to a customer of theirs, which may be irrelevant and inappropriate for an SCSP to know.

This does not mean that data from the Sessions and Power Regulation modules cannot be joined in one system, or that a certain Party cannot play the eMSP and SCSP roles at the same time. This is still *possible* with OCPI 3.0; it is just not *required*.

The definition of the MeterSample object is probably incomplete. It was written without detailed input from SCSPs or CPOs with good experience doing Smart Charging. The following are examples of things that are currently not included because we're not sure if these should be included, and if so, how they should be:

- Voltage measurements, and
- Energy delivery over a certain time period (that is transferring a period start timestamp, period end timestamp and energy quantity).

We are very curious how parties with more experience in Smart Charging appreciate the perspective shift that OCPI 3.0 tries to introduce here, and if they do, which extra possibilities they would need in the MeterSample object to make it work optimally for them.

13.2. Smart Charging Topologies

There are different Smart Charging Topologies possible. Which topology can be used depends on the contracts between different parties.

NOTE

Care has to be taken to prevent mixing the different topologies. When multiple parties start sending Charging Profiles, the resulting charging power might be unpredictable. In case of OCPP Charging Stations, the result will be the minimum of all the Charging Profiles, resulting in a slower than needed charging power.

13.2.1. The eMSP generates Charging Profiles.

The most straight forward topology, the eMSP generates Charging Profiles for its own Drivers, no SCSP is involved. The eMSP holds the relation to the Driver, so if the eMSP knows that its Driver agrees with the eMSP manipulating the charging power, the eMSP is free to do this.

Figure 49. Smart Charging Topology: The eMSP generates Charging Profiles.

OCPI Module Role	Business Role
MeterSample sender	CPO
MeterSample receiver	eMSP

13.2.2. The eMSP delegated Smart Charging to SCSP.

In this topology, the eMSP has delegated the generation of Charging Profiles to an SCSP. For this, the eMSP and SCSP have agreed to use OCPI as the interface.

The eMSP holds the relation to the Driver, so if the eMSP knows that its Driver agrees with the eMSP manipulating the charging power, the eMSP is free to do this. The eMSP can forward OCPI Session, Location and/or MeterSample objects to the SCSP. The SCSP can act on the received/updated objects, by sending Charging Profiles via the eMSP to the CPO.

The eMSP and SCSP have to take into account that they have to oblige to local privacy laws when exchanging information about eMSPs' Drivers.

From the CPO point of view, this topology is similar to the one above, the CPO will not know the difference.

Figure 50. Smart Charging Topology: The eMSP generates Charging Profiles.

Connection	OCPI Module Role	Business Role
SCSP - eMSP	MeterSample receiver	SCSP
SCSP - eMSP	MeterSample sender	eMSP
eMSP - CPO	MeterSample receiver	eMSP
eMSP - CPO	MeterSample sender	CPO

13.2.3. The CPO delegated Smart Charging to SCSP.

In this topology, the CPO has delegated the generation of Charging Profiles to a SCSP. For this, the CPO and SCSP have agreed to use OCPI as the interface.

The CPO holds a relation to the EVSE on which charging happens, or to the owner of said EVSE. As the CPO does not have a direct relation with the Drivers, the CPO needs to make sure the EV driver knows that the charging power might not be the maximum the driver has expected, this could be something as simple as a sticker on the Charging Station, or might even be part of the tariff text.

The CPO might generate Charging Profiles themselves, but as OCPI is then not used this is not part of this document.

The CPO can forward OCPI Location, Session and/or MeterSample objects to the SCSP. the SCSP can act on the received/updated objects, by sending Charging Profiles to the CPO.

The CPO and SCSP have to take into account that they have to oblige to local privacy laws when exchanging information about eMSPs' Drivers.

In this topology, the eMSP is not aware that the CPO is using OCPI to receive Charging Profiles from the SCSP.

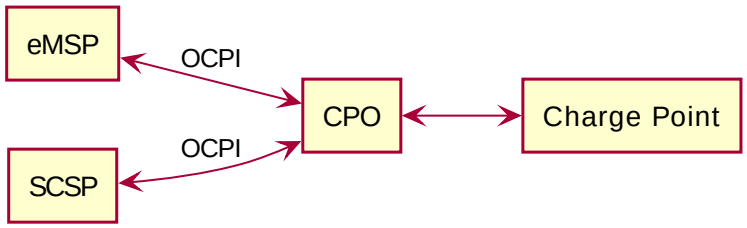


Figure 51. Smart Charging Topology: The eMSP generates Charging Profiles.

OCPI Module Role	Business Role
MeterSample receiver	SCSP
MeterSample sender	CPO

13.3. Changes from OCPI 2.2.1

- The module has gained functionalities for the SCSP or eMSP to receive feedback about the effect of the profiles that they sent.
- The module was renamed from Charging Profiles to Power Regulation accordingly.
- The module now allows SCSPs and eMSPs to set Charging Profiles on groupings of EVSEs, in addition to allowing to set Profiles on Charging Sessions.

- The module now allows SCSPs and eMSPs to set default Charging Profiles on EVSEs.
- The functionality of pushing the active Charging Profile was removed. As an alternative, consider use cases Replicate MeterSample objects from one Party to another Party or Notify Party of Active Charging Profile.

13.4. Replicating MeterSample objects

13.4.1. UC: 13.01 - Replicate MeterSample objects from one Party to another Party

1	**Objective(s)**	**1.** Party X on a Platform A obtains a meter readings data point from Party Y's charging infrastructure.
2	**Description**	Using the use cases of the Party-Issued Objects chapter, MeterSample objects are replicated from Party Y to Party X
3	**Actors**	SCSP, CPO, eMSP, Hub
4	**Flow**	1. Party X subscribes to Party Y's Power Regulation module 2. Party X sets one or more Charging Profiles on Party Y's EVSEs and/or Charging Sessions. 3. Party Y pushes meter reradings relevant to Party X for regulation, as soon as Party Y receives these readings from the charging devices. 4. This continues until either party cancels the subscription, or Party X stops setting Charging Profiles on Party Y's EVSEs and/or Charging Sessions.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Power Regulation to Party X on Platform A.
6	**Postconditions**	Party X has accurate, near realtime knowledge of the power of EVSEs and/or Charging Sessions that it set Charging Profiles on.
7	**Error reporting**	Error reporting happens according to the use cases in the Party-Issued Objects use cases.
8	**Remark(s)**	

Table 50. UC: 13.01 Requirements

ID	Precondition	Requirement
R.13.01.01		Platform A SHALL subscribe according to use case Subscribe to the Party Issued Objects of a certain Module of a certain Party, using "metersamples" as the ModuleID value.
R.13.01.02		Platforms A and B MAY also follow use cases Subscribe to Party Issued Objects of a certain Module of a certain Party as a Hub, Subscribe to Party Issued Objects of a certain Module of a Hub or Subscribe to Party Issued Objects of a certain Module of a Hub as a Hub while subscribing according to R.13.01.01

ID	Precondition	Requirement
R.13.01.03	Platform B sends a Party Issued Object update to Party X on Platform A according to use case Send a full update of a Party Issued Object to a Subscribed Platform in the context of the subscription created according to R.13.01.01	Platform B SHOULD set the value of the "payload" field of the PartyIssuedObjectUpdate object in the request body to a JSON object that conforms to the MeterSample object type.

13.5. Remote Procedure Calls for Power Regulation

13.5.1. UC: 13.02 - Set a Charging Profile on a grouping of EVSEs

1	**Objective(s)**	Party Y applies a Charging Profile to a EVSE or a group of EVSEs as instructed by Party X
2	**Description**	An asynchronous remote procedure call is made by Party X to Party Y. In the request Party X sends the Charging Profile that they wish to apply and the EVSEs to which they wish it to be applied. In the response Party Y sends a confirmation or a rejection of the Charging Profile application.
3	**Actors**	SCSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the Charging Profile and the IDs of the EVSEs to apply it to. 2. Party Y decides if it accepts this Charging Profile for application to these EVSEs. 3. If Party Y accepts, it applies the Charging Profile to these EVSEs and informs Party X that it was applied. 4. If Party Y does not accept, it informs Party X that the Charging Profile is not applied.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Power Regulation to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to apply a Charging Profile to one or more of Party Y's EVSEs * Party Y applied the Charging Profile to the EVSEs mentioned in Party X's request. Also Party X knows that their request was turned into action and that they can expect to receive Meter Samples about the realised energy consumption at those EVSEs.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

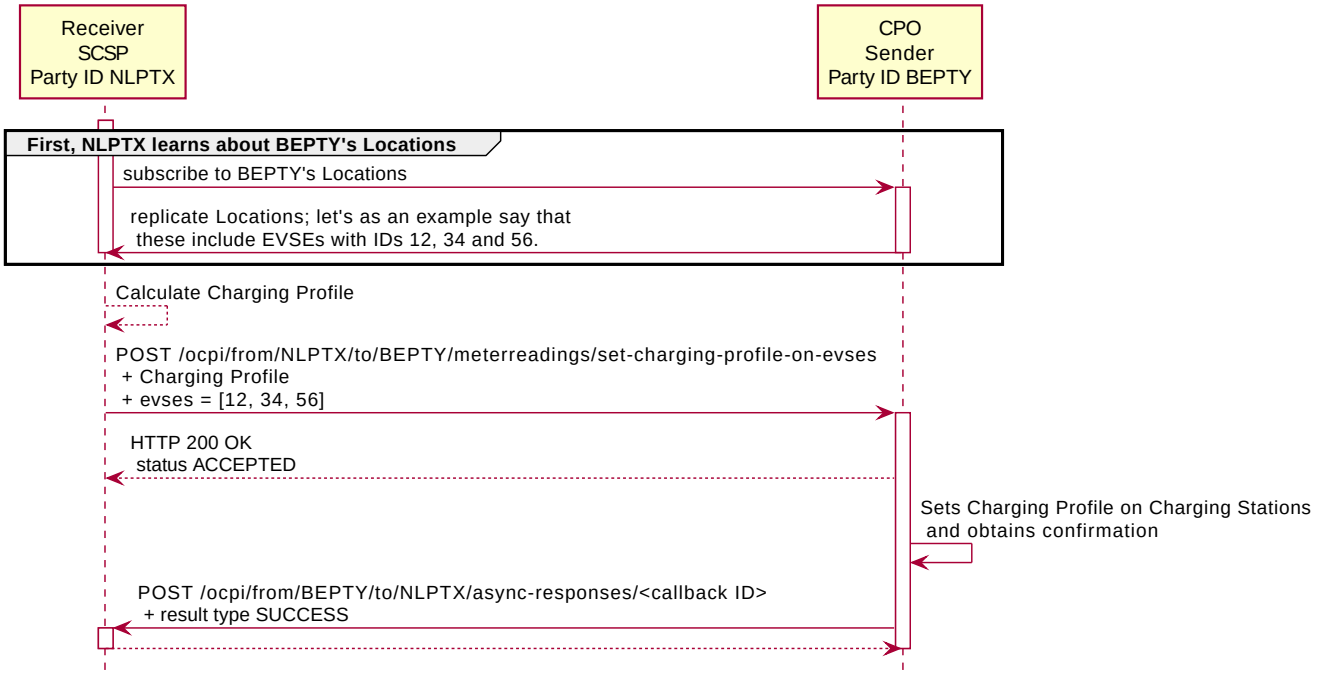


Figure 52. Sequence Diagram: Set a Charging Profile on a grouping of EVSEs

Table 51. UC: 13.02 Requirements

ID	Precondition	Requirement
R.13.02.01		Platform A SHALL make the request to apply a Charging Profile to one or more EVSEs following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.02.02		Platform A SHALL use "set-charging-profile-on-evses" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.02.03		Platform A SHALL use a SetChargingProfileOnEvsesRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.02.04	Party Y cannot fulfill the request from Party X because of the state of one or more of the EVSEs	Platform B SHALL respond to Platform A's request with ACCEPTED in the `data` field of the OcpiResponse in the response body. Platform B SHALL then send an asynchronous response with the result_type field of the AsyncResponse set to REJECTED and the error field of the AsyncResponse set to an appropriate RegulationError value.
R.13.02.05		Party X MUST provide a start_date_time in the Charging Profile when setting a Charging Profile with this use case. This is because there is no Charging Session context relative to which time cutoffs can be interpreted.
R.13.02.06	Party Y is willing and able to apply the Charging Profile requested by Party X.	Party Y SHOULD make sure that the total combined energy delivery of all EVSEs mentioned in Party X's requested does not exceed the limit of the applicable ChargingProfilePeriod at any time.

ID	Precondition	Requirement
R.13.02.07	Party Y applied the Charging Profile to the EVSEs as requested by Party X	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse set to SUCCESS and the `error` and `payload` fields of the AsyncResponse both left unset.
R.13.02.08	Party Y applied the Charging Profile to the EVSEs as requested by Party X and Party X is subscribed to Party Y's Power Regulation.	Party Y SHOULD issue Meter Sample objects to Party X so that Party X can track the execution of the Charging Profile.
R.13.02.09		The `numberPhases` and `phaseToUse` fields SHALL be unset in all ChargingProfilePeriod objects in the charging profile in the payload of Platform A's request.

NOTE

R.13.02.08 is rather vague in that it does not specify exactly which measurands Party Y should share with Party X, and in which units and at what times it should do so. This is deliberate; there is little certainty at the time of writing on how this monitoring will work exactly and how much freedom Party Y will have to modify the way it happens. Therefore OCPI leaves this open for the parties to agree outside of the protocol messages.

13.5.2. UC: 13.03 - Set a Charging Profile on a Charging Session

1	**Objective(s)**	Party Y applies a Charging Profile to a Charging Session as instructed by Party X
2	**Description**	An asynchronous remote procedure call is made by Party X to Party Y. In the request Party X sends the Charging Profile that they wish to apply and the Charging Session to which they wish it to be applied. In the response Party Y sends a confirmation or a rejection of the Charging Profile application.
3	**Actors**	eMSP, SCSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the Charging Profile and the ID of the Charging Session to apply it to. 2. Party Y decides if it accepts this Charging Profile for application to these EVSEs. 3. If Party Y accepts, it applies the Charging Profile to these EVSEs and informs Party X that it was applied. 4. If Party Y does not accept, it informs Party X that the Charging Profile is not applied.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Power Regulation to Party X on Platform A.

6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to apply a Charging Profile to one of Party Y's Sessions; or * Party Y applied the Charging Profile to the Charging Session mentioned in Party X's request. Also Party X knows that their request was turned into action and that they can expect to receive Meter Samples about the realised energy consumption in that Charging Session.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

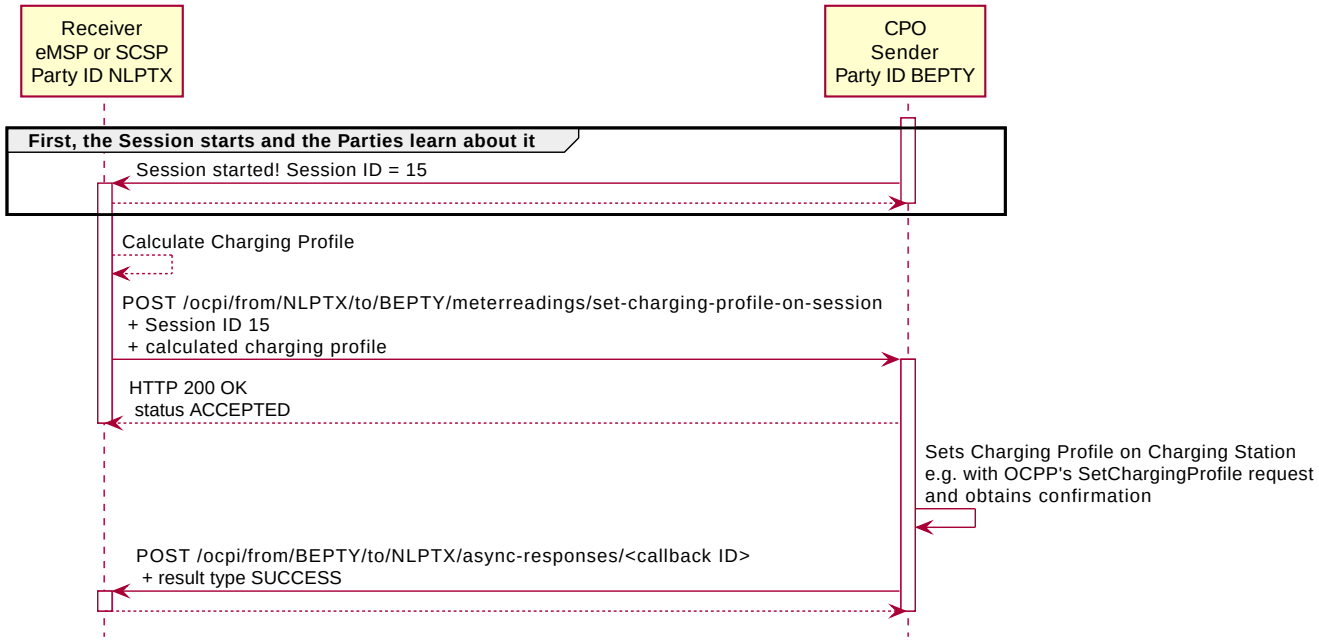


Figure 53. Sequence Diagram: Set a Charging Profile on a Charging Session

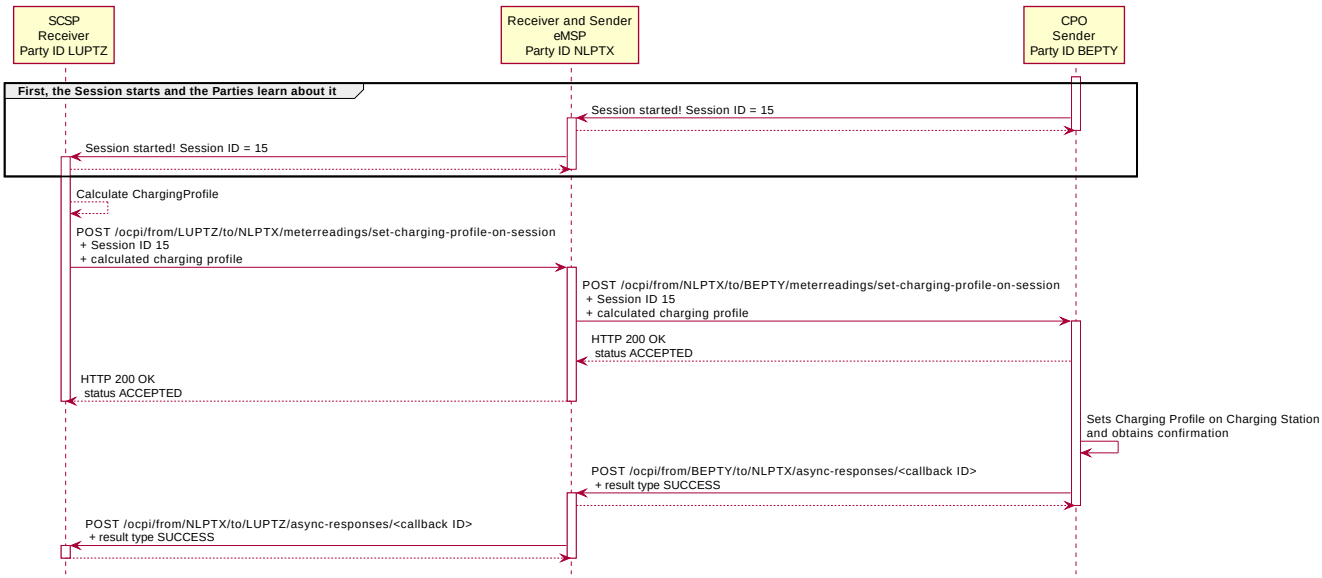


Figure 54. Sequence Diagram specifically for the topology where eMSP delegates Smart Charging to an SCSP.

Table 52. UC: 13.03 Requirements

ID	Precondition	Requirement
R.13.03.01		Platform A SHALL make the request to set a Charging Profile on a Charging Session following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.03.02		Platform A SHALL use "set-charging-profile-on-session" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.03.03		Platform A SHALL use a SetChargingProfileOnSessionRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.03.04	Party Y cannot fulfill the request from Party X because of the state of the device that the Session is happening on.	Platform B SHALL respond to Platform A's request with ACCEPTED in the `data` field of the OcpiResponse in the response body. Platform B SHALL then send an asynchronous response with the result_type field of the AsyncResponse set to REJECTED and the error field of the AsyncResponse set to an appropriate RegulationError value.
R.13.03.05	Party Y is willing and able to apply the Charging Profile requested by Party X.	Party Y SHOULD make sure that the rate of energy delivery in the Charging Session mentioned in Party X's requested does not exceed the limit of the applicable ChargingProfilePeriod at any time.
R.13.03.06	Party Y applied the Charging Profile to the Charging Session as requested by Party X	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse set to SUCCESS and the `error` and `payload` fields of the AsyncResponse both left unset.
R.13.03.07	Party Y applied the Charging Profile to the Charging Session as requested by Party X and Party X is subscribed to Party Y's Power Regulation.	Party Y SHOULD issue Meter Sample objects to Party X so that Party X can track the execution of the Charging Profile.

NOTE

R.13.03.07 is rather vague in that it does not specify exactly which measurands Party Y should share with Party X, and in which units and at what times it should do so. This is deliberate; there is little certainty at the time of writing on how this monitoring will work exactly and how much freedom Party Y will have to modify the way it happens. Therefore OCPI leaves this open for the parties to agree outside of the protocol messages.

13.5.3. UC: 13.04 - Set Default Charging Profile

1	**Objective(s)**	Party Y applies a Charging Profile sent by Party X to Charging Sessions that are newly started on a certain subset of Party Y's EVSEs.

2	**Description**	An asynchronous remote procedure call is made by Party X to Party Y. In the request Party X sends the Charging Profile that they wish to apply and the EVSEs to which they wish it to be applied for newly starting Charging Sessions. In the response Party Y sends a confirmation or a rejection of the Charging Profile application.
3	**Actors**	eMSP, SCSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the Charging Profile and the IDs of the EVSEs to apply it to. 2. Party Y decides if it accepts this Charging Profile for application to new Charging Sessions on these EVSEs. 3. Party Y asynchronously informs Party X if the Charging Profile was applied as requested or not.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Power Regulation to Party X on Platform A.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their attempt to apply a Charging Profile to newly starting Charging Sessions on one or more of Party Y's EVSEs * Party Y stored the Charging Profile and will apply it to newly starting Charging Sessions. Also Party X knows that their request was turned into action and that they can expect to receive Meter Samples about the realised energy consumption in Charging Sessions that are started on the EVSEs that Party X listed in the requested.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	Despite the naming, CPOs are **not** required to implement this use case by sending the profile given by Party X to the charging station as an OCPP `TxDefaultProfile`. CPOs are free to set the `TxDefaultProfile` to a different profile that assures safety of the electrical circuit. CPOs that do so can still implement OCPI's "Set Default Charging Profile" use case by sending a TxProfile to the Charging Station once the CPO's Charging Station Management System has learned about the Charging Session and has done additional checks.

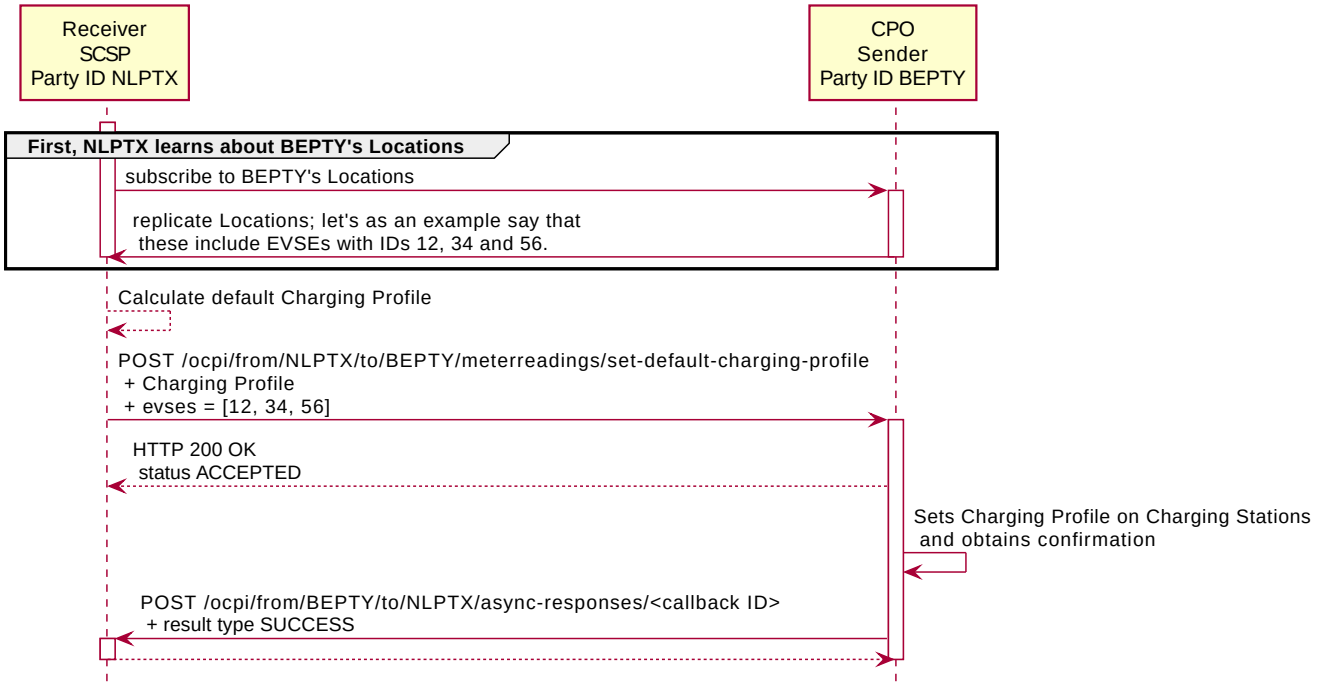


Figure 55. Sequence Diagram: Set Default Charging Profile

Table 53. UC: 13.04 Requirements

ID	Precondition	Requirement
R.13.04.01		Platform A SHALL make the request to apply a default Charging Profile to one or more EVSEs following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.04.02		Platform A SHALL use "set-default-charging-profile" the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.04.03		Platform A SHALL use a SetChargingProfileOnEvsesRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.04.04	Party Y cannot fulfill the request from Party X because of the state of one or more of the EVSEs	Platform B SHALL respond to Platform A's request with ACCEPTED in the `data` field of the OcpiResponse in the response body. Platform B SHALL then send an asynchronous response with the result_type field of the AsyncResponse set to REJECTED and the error field of the AsyncResponse set to an appropriate RegulationError value.
R.13.04.05	Party Y is willing and able to apply the Charging Profile requested by Party X.	Whenever a Charging Session is started on an EVSE mentioned in the request, Party Y SHOULD make sure that the power with which the EV is charging in that session does not exceed the limit of the applicable ChargingProfilePeriod at any time.

ID	Precondition	Requirement
R.13.04.06	Party Y applied the default Charging Profile to the EVSEs as requested by Party X	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse set to SUCCESS and the `error` and `payload` fields of the AsyncResponse both left unset.
R.13.04.07	Party Y applied the Charging Profile to the EVSEs as requested by Party X and Party X is subscribed to Party Y's Power Regulation.	Party Y SHOULD issue Meter Sample objects to Party X so that Party X can track the execution of the Charging Profile.

13.5.4. UC: 13.05 - Get Active Charging Profile

1	**Objective(s)**	Party X learns from Party Y how Party Y plans to execute a Charging Profile that was previously set by Party X.
2	**Description**	An asynchronous remote procedure call is made by Party X to Party Y. In the request Party X sends the ID of the Charging Profile that they are inquiring about. In the response Party Y sends a confirmation or a rejection of the request. If the request is accepted, Party Y then sends the Active Charging Profile to Party X in the asynchronous response.
3	**Actors**	eMSP, SCSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the ID of the Charging Profile that Party X wants to receive the Active Charging Profile for. 2. Party Y decides if it accepts this request for an Active Charging Profile. 3. If Party Y accepts, it gathers the information comprising the Active Chrging Profile and sends it to Party X. 4. If Party Y does not accept, it informs Party X that it will not give Party X the Active Charging Profile.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Power Regulation to Party X on Platform A. Party X previously set a Charging Profile on Party Y's systems using Set a Charging Profile on a Charging Session or Set a Charging Profile on a Grouping of EVSEs.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their request for an Active Charging Profile; or * Party X knows the Active Chraging Profile that Party Y is using.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

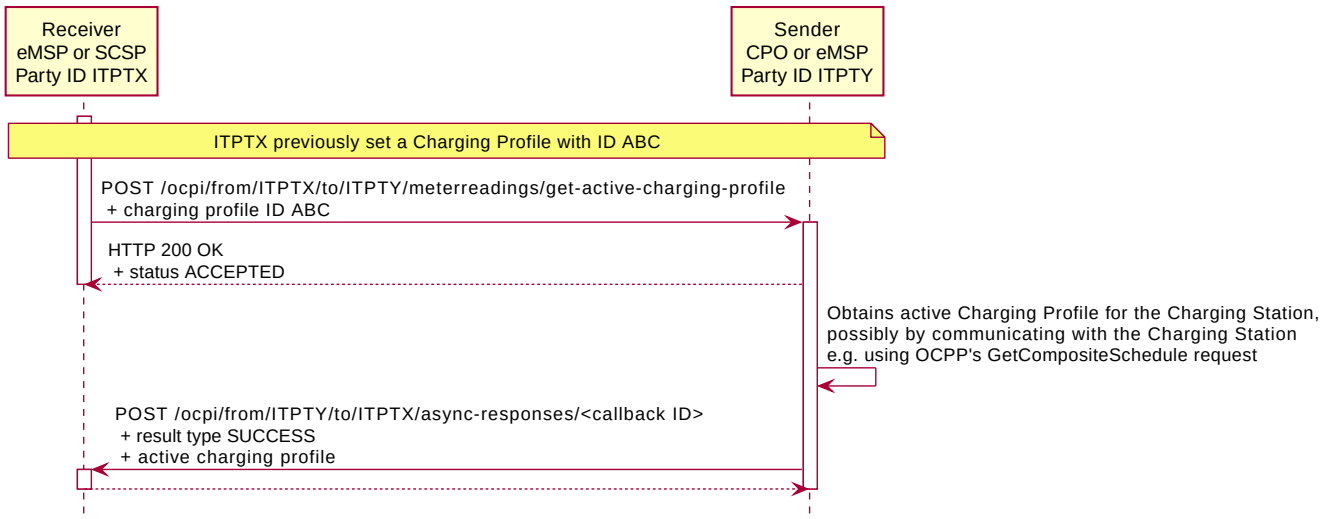


Figure 56. Sequence Diagram: Get Active Charging Profile

Table 54. UC: 13.05 Requirements

ID	Precondition	Requirement
R.13.05.01		Platform A SHALL make the request to get an Active Charging Profile following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.05.02		Platform A SHALL use "get-active-charging-profile" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.05.03		Platform A SHALL use a SetChargingProfileOnSessionRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.05.04	Party Y cannot fulfill the request from Party X because of the state of one or more of the devices that the Charging Profile is executed on.	Platform B SHALL respond to Platform A's request with ACCEPTED in the `data` field of the OcpiResponse in the response body. Platform B SHALL then send an asynchronous response with the result_type field of the AsyncResponse set to REJECTED and the error field of the AsyncResponse set to an appropriate RegulationError value.
R.13.05.05	Party Y is willing and able to provide the Active Charging Profile requested by Party X.	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse set to SUCCESS and the `error` field of the AsyncResponse left unset and the `payload` field of the AsyncResponse set to a ActiveChargingProfile object that fulfills Party X's request. If there is no longer any Charging Profile active, the ActiveChargingProfile SHALL reflect this by showing the maximum charging capacity of the EVSE.

NOTE

When choosing the `duration` value in the GetActiveChargingProfile request object, Party X has to balance the duration between maximizing the information gained and the data usage and computation to execute on the request. Consider that asking for longer duration than necessary

might result in additional data costs, while its added value diminishes with every change in the schedule.

13.5.5. UC: 13.06 - Clear Charging Profile

1	**Objective(s)**	Party X requests Party Y to no longer take into account a Charging Profile that Party X previously requested Party Y to apply.
2	**Description**	An asynchronous remote procedure call is made by Party X to Party Y. In the request Party X sends the ID of the Charging Profile that they would like to clear. In the response Party Y sends a confirmation or a rejection of the request. If the request is accepted, Party Y will adjust the rate of energy delivery on their infrastructure so that the cleared Charging Profile is no longer applied.
3	**Actors**	eMSP, SCSP, CPO
4	**Flow**	1. Platform A makes a request on behalf of Party X to Platform B receiving the request on behalf of Party Y. This request contains the ID of the Charging Profile that Party X wants to clear. 2. Party Y decides if it accepts this request to clear a Charging Profile. 3. If Party Y accepts, it clears the Charging Profile and asynchronously responds with a confirmation to Party X. 4. If Party Y does not accept, it informs Party X that it will not clear the Charging Profile.
5	**Preconditions**	Both platforms have set up an OCPI connection with Handshake OCPI Connection Parameters. Platform B serves Party Y to Party X. Platform A serves Party X to Party Y. Platform B serves Party Y's Power Regulation to Party X on Platform A. Party X previously set a Charging Profile on Party Y's systems using Set a Charging Profile on a Charging Session or Set a Charging Profile on a Grouping of EVSEs.
6	**Postconditions**	One of these two: * Party X knows Party Y rejected their request to clear the Charging Profile; or * Party Y stopped applying the Charging Profile and Party X is aware of this.
7	**Error handling**	Error reporting by Platform B follows the generic mechanism described in Make a Request to a Party on behalf of a Party.
8	**Remark(s)**	

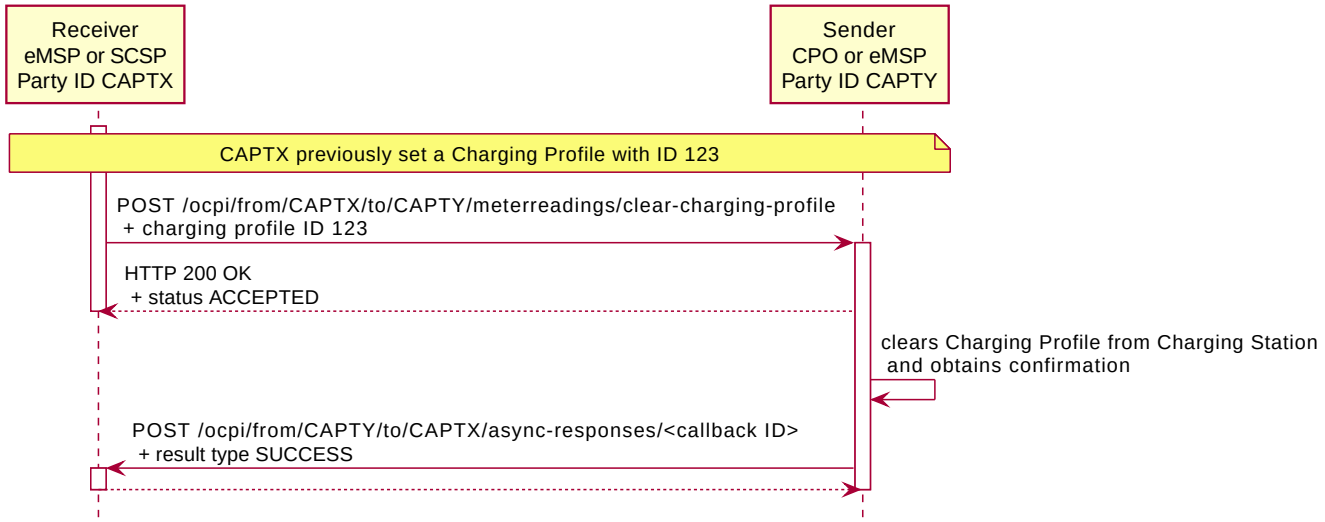


Figure 57. Sequence Diagram: Clear Charging Profile

Table 55. UC: 13.06 Requirements

ID	Precondition	Requirement
R.13.06.01		Platform A SHALL make the request to clear a Charging Profile following Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.06.02		Platform A SHALL use "clear-charging-profile" as the operation name for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.06.03		Platform A SHALL use a ClearChargingProfileRequest in the payload field of the request body for Make a Remote Procedure Call Allowing Asynchronous Responses.
R.13.06.04	Party Y cannot fulfill the request from Party X because of the state of one or more of the devices that the Charging Profile is executed on.	Platform B SHALL respond to Platform A's request with ACCEPTED in the `data` field of the OcpiResponse in the response body. Platform B SHALL then send an asynchronous response with the result_type field of the AsyncResponse set to REJECTED and the error field of the AsyncResponse set to an appropriate RegulationError value.
R.13.06.05	Party Y is willing and able to clear the Charging Profile requested by Party X.	Platform B SHALL send an asynchronous response with the result_type field of the AsyncResponse set to SUCCESS and the `error` and `payload` fields of the AsyncResponse left unset.

13.6. Data types

13.6.1. ActiveChargingProfile *class*

Property	Type	Card.	Description
start_date_time	DateTime	1	Date and time at which the Charge Point has calculated this ActiveChargingProfile. All time measurements within the profile are relative to this timestamp.
charging_profile	ChargingProfile	1	Charging Profile structure defines a list of charging periods.

13.6.2. ChargingRateUnit *enum*

Unit in which a Charging Profile is defined.

Value	Description
W	Watts (power) This is the TOTAL allowed charging power. If used for AC Charging, the phase current should be calculated via: Current per phase = Power / (Line Voltage * Number of Phases). The "Line Voltage" used in the calculation is the Line to Neutral Voltage (VLN). In Europe and Asia VLN is typically 220V or 230V and the corresponding Line to Line Voltage (VLL) is 380V and 400V. The "Number of Phases" is the numberPhases from the ChargingProfilePeriod. It is usually more convenient to use this for DC charging. Note that if numberPhases in a ChargingProfilePeriod is absent, 3 SHALL be assumed.
A	Amperes (current) The amount of Ampere per phase, not the sum of all phases. It is usually more convenient to use this for AC charging.

13.6.3. Unit *OpenEnum*

Unit in which a MeterReading is reported.

Value	Description
A	Amperes
Hz	Hertz
PERCENT	Percentage
W	Watts

13.6.4. ChargingProfile *class*

A Charging Profile consists of a list of charging profile periods.

Property	Type	Card.	Description
id	CiAsciiString[1..36]	1	An ID of the Charging Profile that is requested to be set. This serves to refer to the Profile later to clear it or to inquire about the Active Charging Profile.

Property	Type	Card.	Description
start_date_time	DateTime	?	Starting point of an absolute profile. If absent the profile will be relative to start of charging.
duration	int	?	Duration of the Charging Profile in seconds. If the duration is left empty, the last period will continue indefinitely or until end of the transaction in case start_date_time is absent.
charging_rate_unit	ChargingRateUnit	1	The unit of measure.
min_charging_rate	number	?	Minimum charging rate supported by the EV. The unit of measure is defined by the chargingRateUnit. This parameter is intended to be used by a local smart charging algorithm to optimize the power allocation for in the case a charging process is inefficient at lower charging rates. Accepts at most one digit fraction (e.g. 8.1)
charging_profile_period	ChargingProfilePeriod	*	List of ChargingProfilePeriod elements defining maximum power or current usage over time.

13.6.5. ChargingProfilePeriod *class*

The ChargingProfilePeriod data structure defines a time period in a Charging Profile, as used in: ChargingProfile

Property	Type	Card.	Description
start_period	int	1	Start of the period, in seconds from the start of profile. The value of StartPeriod also defines the stop time of the previous period.
limit	number	1	Charging rate limit during the profile period, in the applicable chargingRateUnit, for example in Amperes (A) or Watts (W). Accepts at most one digit fraction (e.g. 8.1).
numberPhases	number	?	The number of AC phases that the CPO can use for charging. For profiles applying to DC charging stations, no value should be given. For AC charging stations, the CPO should assume a default value of 3 when using a ChargingProfilePeriod in which this value is not given.
phaseToUse	number	?	Which AC phase the CPO should use to charge, if numberPhases is set to 1. The only possible values are 1, 2 and 3.

13.6.6. ComponentLevel *enum*

Value	Description
EV	The meter reading applies to the Electric Vehicle (EV) connected to the EVSE.
EVSE	The meter reading applies to a single EVSE.
GROUP	The meter reading applies to a grouping of EVSEs.

13.6.7. ComponentLocation *OpenEnum*

Describes at which point in the power circuit, relative to the component, a MeterReading was obtained.

Value	Description
INLET	The meter reading was obtained at the inlet, that is, the meter reading describes the power that is taken from the power supply by the component.
OUTLET	The meter reading was obtained at the outlet, that is, the meter reading describes the power being delivered by the component.

13.6.8. GetActiveChargingProfileRequest *class*

Parameters for a Get Active Charging Profile request.

Parameter	Type	Card.	Description
charging_profile_id	CiAsciiString[1..36]	1	The unique ID that identifies the Charging Profile for which an Active Charging Profile is requested to the CPO.
duration	int	1	Length of the requested ActiveChargingProfile in seconds.

13.6.9. Measurand *OpenEnum*

That what is measured in a MeterReading.

Value	Description
CURRENT	Electric current, typically reported in Amperes.
POWER	The electric power currently being consumed, typically reported in Watts.
FREQUENCY	The frequency of the AC current alternation.
SOC	The state of charge, typically reported as a percentage.

13.6.10. MeterReading *class*

A record of a measurement of a quantity relevant to regulation of charging power.

Property	Type	Card.	Description
value	number	1	The measured value, in the unit given by the `unit` field.
measurand	Measurand	1	The quantity that was measured.
unit	Unit	1	The unit in which this meter reading is given.
component_level	ComponentLevel	1	The level of grouping for which the meter reading is given.
location	ComponentLocation	1	At which point in the energy flow relative to the component the meter reading was obtained.

Property	Type	Card.	Description
phase	Phase	?	Which phase the reading applies to. When this field is not given, the measured value is interpreted as an overall value.

13.6.11. MeterSample *class*

A record of one or more measurements that are made at the same time.

Transferring MeterSample objects is the main mechanism by which OCPI allows Parties who control charging power at other Parties' infrastructure to see what the impact of their control is.

Property	Type	Card.	Description
timestamp	DateTime	1	The time at which the measurements were taken
evse_id	CiAsciiString[1..36]	?	The UID of the EVSE to which these measurements apply, if any.
session_id	CiAsciiString[1..36]	?	The ID of the Charging Session to which these measurements apply, if any.
charging_profile_id	CiAsciiString[1..36]	?	The Charging Profile for which these measurements are sent.
readings	MeterReading	+	The measurements

13.6.12. Phase *enum*

Indicates to which phase or phases of the power supply a meter reading applies.

Value	Description
L1	Measured on L1
L2	Measured on L2
L3	Measured on L3
N	Measured on Neutral
L1-N	Measured on L1 with respect to Neutral conductor
L2-N	Measured on L2 with respect to Neutral conductor
L3-N	Measured on L3 with respect to Neutral conductor
L1-L2	Measured between L1 and L2
L2-L3	Measured between L2 and L3
L3-L1	Measured between L3 and L1

13.6.13. RegulationError *enum*

An enumeration to report reasons for failure to apply a Charging Profile.

Value	Description
TOO_OFTEN	The Charging Profile request was rejected by the CPO because requests are sent more often than the CPO allows.
UNKNOWN_SESSION	The Session in the request command is not known by this CPO.
UNKNOWN_EVSE	An EVSE mentioned in the request is not known by this CPO.

13.6.14. SetChargingProfileOnEvsesRequest *class*

Property	Type	Card.	Description
charging_profile	ChargingProfile	?	The Charging Profile to apply to the group of EVSEs. If this field is unset, it indicates that a previously set Charging Profile is to be removed.
evses	CiAsciiString[1..36]	+	UIDs of the EVSEs to apply the Charging Profile to.

13.6.15. SetChargingProfileOnSessionRequest *class*

Property	Type	Card.	Description
charging_profile	ChargingProfile	1	The Charging Profile to apply to the group of EVSEs
session_id	CiAsciiString[1..36]	1	ID of the Charging Session to apply the Charging Profile to

13.6.16. ClearChargingProfileRequest *class*

Property	Type	Card.	Description
charging_profile_id	CiAsciiString[1..36]	yes	The unique ID that identifies the Charging Profile that is to be cleared.