

Processes for Extension Development for OCPI

Reinier Lamers

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

1.0, 2026-09-10: Accepted

Table of Contents

1. Motivation . . . 2
2. Domain . . . 2
  2.1. Terminology . . . 2
  2.2. Use Cases . . . 2
    2.2.1. Use Case 1: As an OCPI Platform, I want to support new use cases for my stakeholders not foreseen in existing OCPI versions . . . 2
    2.2.2. Use Case 2: As an OCPI Platform, I want to support new technologies not foreseen in existing OCPI versions . . . 3
    2.2.3. Use Case 3: As the EV Roaming Foundation, we want to help OCPI Platforms work around protocol flaws in existing OCPI versions . . . 3
    2.2.4. Use Case 4: As an OCPI Platform, I want to be interoperable with other OCPI Platforms, regardless of which extensions they do or do not support. . . . 3
  2.3. The Standard Extension Development Process . . . 3
    2.3.1. Rationale . . . 3
      2.3.1.1. IP of standard extensions . . . 3
      2.3.1.2. The risks of extensibility . . . 4
    2.3.2. Process rules . . . 4
      2.3.2.1. Extension document lifecycle transitions . . . 5
      2.3.2.2. Process tools and visibility . . . 6
    2.3.3. Incompatible changes . . . 7
  2.4. Extension points . . . 7
    2.4.1. Open enumerations . . . 8
    2.4.2. JSON Object fields . . . 8
    2.4.3. Party-Issued Objects . . . 8
    2.4.4. RPC endpoints . . . 8
    2.4.5. Types . . . 9
    2.4.6. HTTP endpoints . . . 9
  2.5. Invariants . . . 9
    2.5.1. Pseudo-invariants . . . 10
  2.6. Hubs and other "middleboxes" . . . 10
    2.6.1. Requirements for middleboxes handling open enumerations . . . 10
    2.6.2. Requirements for middleboxes handling JSON object fields . . . 10
    2.6.3. Requirements for middleboxes handling Party Issued Objects . . . 11
    2.6.4. Requirements for middleboxes handling RPC endpoints . . . 11
    2.6.5. Requirements for middleboxes handling types . . . 11
    2.6.6. Requirements for middleboxes handling HTTP endpoints . . . 11
3. Extension discovery . . . 11
  3.1. Extension discovery in OCPI 3.0 . . . 11
    3.1.1. Discovering extensions supported across the whole Platform . . . 12
    3.1.2. Discovering extensions supported only for specific Parties . . . 12
    3.1.3. Versioning . . . 12
  3.2. Extensibility discovery and negotiation in OCPI 2.3.0 . . . 13
    3.2.1. Discovering and negotiating extensions supported across the whole Platform . . . 13
    3.2.2. Discovering and negotiating extensions supported only for specific Parties . . . 14
    3.2.3. Versioning . . . 14

4. Actions to be taken by the OCPI editors to foster the extension development process 14
5. Actions to be recommended by the OCPI Development Working Group to the EV Roaming Foundation Board to foster the extension development process 14
6. Actions to be recommended by the OCPI Development Working Group to the EV Roaming Foundation Advisory Board to foster the extension document process 15
7. References 15

Copyright © 2014 – 2026 EVRoaming Foundation. All rights reserved.

This document is made available under the *Creative Commons Attribution- NoDerivatives 4.0 International Public License*

(https://creativecommons.org/licenses/by-nd/4.0/legalcode).

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.

1. Motivation

The EV charging industry is slow to adopt new releases of its own communication protocols. As of December 2025, most charging stations use OCPP 1.6, which was released in 2015, to communicate with their management system. Thereby they forego the improvements provided by OCPP 2.0.1 (released in 2020) and OCPP 2.1 (released in 2025). Where different EV charging operators and providers communicate among each other, they typically use OCPI 2.1.1 (released in 2017), foregoing the improvements provided by OCPI 2.2.1 (released in 2021) or OCPI 2.3.0 (released in 2025).

This lag in adoption of new standards means that protocols created by standardization committees today will only come into use some five years later, if at all. It means that EV charging operators and providers have to work with tools that are at least five years old while they are addressing new, urgent challenges in the industry. Among the mounting challenges that will be impossible to solve without implementing newer standards are increasing cybersecurity awareness, a push for transparent charging pricing, and the increasing adoption of electric vehicles by heavy-duty fleet operators.

This document aims to speed up adoption of improved versions of OCPI by describing how improvements can be introduced to OCPI without creating the barriers to adoption that plagued earlier versions of OCPI and OCPP. The document aims to work towards a future in which OCPI implementers can easily add new capabilities to their systems without breaking interoperability with existing partners.

In this future, OCPI will be both backward compatible and forward compatible. OCPI being backward compatible means that OCPI Platforms developed to support a newer version of OCPI can process output from OCPI Platforms conforming to older versions of OCPI. OCPI being forward compatible means that OCPI Platforms developed to support an older version of OCPI can process output from OCPI Platforms conforming to newer versions of OCPI.

2. Domain

2.1. Terminology

This document uses the terms "custom extension" and "standard extension". By "custom extension" we mean extra fields, endpoints, etcetera that are used in OCPI messages or on OCPI connections. By "standard extension", we mean a document published by the EV Roaming Foundation with extra requirements that can be used in addition to the requirements of a certain OCPI version. This distinguishes ad-hoc additions to OCPI by implementers from additions that come about through the EV Roaming Foundation’s consensus-based processes.

2.2. Use Cases

2.2.1. Use Case 1: As an OCPI Platform, I want to support new use cases for my stakeholders not foreseen in existing OCPI versions

Often, new capabilities in OCPI are needed because OCPI implementers realize that there is a previously unrecognized use case in the EV charging industry that has to be urgently addressed. Examples of this would be the requirement to provide accessible charging facilities, or the need of heavy-duty fleet operators to book charging times and energy delivery in advance.

2.2.2. Use Case 2: As an OCPI Platform, I want to support new technologies not foreseen in existing OCPI versions

New products, services and technologies keep appearing at a steady pace. When this happens, it should be possible for OCPI Platforms to refer to these new technologies in data they exchange via OCPI, without needing to upgrade to an incompatible version of OCPI.

An example of this happening in practice would be the appearance of the Megawatt Charging System (MCS) and SAE J-3400 (NACS) connector types. Another imaginable scenario that falls in this category would be the launch of a new on-site payment system besides the existing ones of Visa, MasterCard etc.

2.2.3. Use Case 3: As the EV Roaming Foundation, we want to help OCPI Platforms work around protocol flaws in existing OCPI versions

It happens that after a version of OCPI is released, the OCPI community realizes that the release fails to support important functionality. In the past, this has led to releases being de facto retracted (like OCPI versions 2.1 and 2.2, and OCPP version 2.0). In other cases, the flaws go unaddressed for years, as happens with OCPI 2.2.1's failure to support loitering fees with grace periods. If it is possible to add functionality to OCPI without breaking compatibility, such oversights can be dealt with much more swiftly and gently.

2.2.4. Use Case 4: As an OCPI Platform, I want to be interoperable with other OCPI Platforms, regardless of which extensions they do or do not support.

OCPI must provide interoperability between Platforms, even when they implement different sets of extensions. Platforms implementing different extension sets should be able to interoperate with the set of extensions implemented by both Platforms.

2.3. The Standard Extension Development Process

2.3.1. Rationale

The goal of extensibility, and extension development, is to allow OCPI Platforms to address new or changed business requirements, while remaining interoperable with Platforms that do not have these capabilities. This section describes the organizational process that we put in place in order to realize the potential of extensibility and avoid its pitfalls.

2.3.1.1. IP of standard extensions

The EVRoaming Foundation offers the OCPI protocol free to download and use by everyone. That also requires that the Intellectual Property (IP) of the protocol and its standard extensions are with EVRoaming Foundation.

In case standard extensions are not fully (from scratch) made by any workgroup of the EVRoaming Foundation, but are based on either complete offered functionality from a company or offered as component/part of functionality by a company, a document is required where parties clearly agree that IP is handed over to EVRF. This does not mean that the concerned company cannot use it anymore themselves, but they are not allowed to claim IP or ask money for usage when used as part of products from the EVRoaming Foundation. The document for handing over IP, can be requested from operations@evroaming.org.

2.3.1.2. The risks of extensibility

In order to achieve our goal of interoperability, we must acknowledge that extensibility also brings risks for interoperability, and we must put processes in place to make sure that extensibility works for interoperability, not against it.

The risks for interoperability that extensibility brings are:

- Interoperability risks: Extensions negate the "integrate once, connect anywhere" promise of a standard. They necessitate synchronized, bilateral implementation between parties, which is rarely scalable across a large network.
- Ecosystem fragmentation: Heavy reliance on extensions fractures the ecosystem. If multiple integrations require unique, non-standard fields to function, OCPI effectively ceases to be a unified standard.
- Maintenance burden: Extensions introduce significant technical debt. They are outside the scope of official compliance testing and may increase the efforts of migrating to future OCPI versions.

To safeguard against these risks, we design a process on the following principles:

- Standard extensions are developed by the EV Roaming Foundation's OCPI Development Working Group. All of the EV Roaming Foundation's Contributors can influence the content of the extension specification before the extension is accepted by the OCPI Development Working Group. Without a consensus in the OCPI Development Working Group, an extension will not be endorsed by the EV Roaming Foundation.
- The requirements of the base specification cannot be altered by a standard extension. Extensions are extensions after all, not modifications.
- The ultimate goal of standard extension development is to make the extension part of a future release of the OCPI base specification. When extensions are included in the OCPI base specification, they also become subject to compliance testing.

The desired outcome of applying these principles is that extension development becomes a way to allow evolution of a unified OCPI specification without breaking interoperability or fragmenting the technical ecosystem.

2.3.2. Process rules

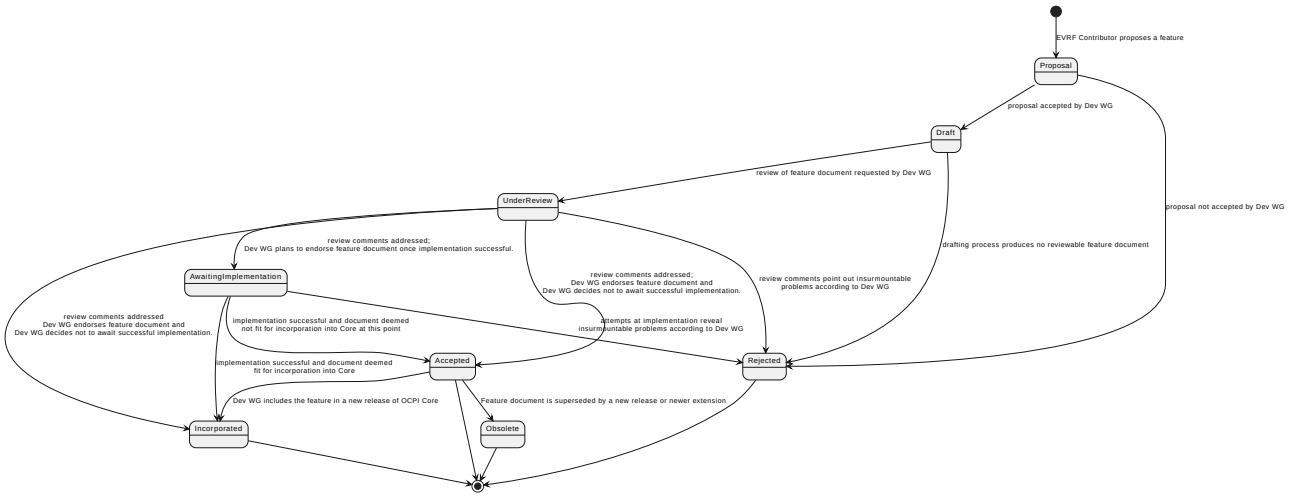
To maintain order and quality when numerous OCPI extensions are developed in parallel, the EV Roaming Foundation will set up processes and document repositories to develop and disseminate extension documents.

Once an extension document is submitted to the EV Roaming Foundation, the EV Roaming Foundation will assign the extension document a number for clear identification. The extension document will then enter into a lifecycle with the following possible stages:

- **proposal**: the EV Roaming Foundation acknowledges that the document was submitted, but has no stance on its merit yet.
- **draft**: the EV Roaming Foundation accepts that the proposal fulfills a business use case. During this phase, EV Roaming Foundation Contributors can propose edits and offer comments. This drafting is not only done by the OCPI Development Working Group, but also by other people from contributors with knowledge of the business and functional aspects.
- **under review**: the EV Roaming Foundation proposes to its Contributors to make the document an EV Roaming Foundation approved specification for use by OCPI platforms, and solicits comments from its Contributors and the general public.

- **awaiting implementation**: after processing all review comments, the EV Roaming Foundation tentatively endorses the document. The EV Roaming Foundation is waiting to make the document accepted or incorporated until actual implementation of the document confirms its technical correctness.
- **rejected**: the document was found by the EV Roaming Foundation's Contributors to not provide a reasonable solution to a business use case. Adherence to the document's requirements is recommended against by the EV Roaming Foundation, and the document is available for historical information only.
- **accepted**: EVRF endorses the document as a specification of how to solve a certain use case.
- **incorporated**: the extension described in the document has been included in a new version of the base OCPI specification. OCPI implementers should consult the new base specification rather than the extension document.
- **obsolete**: the document used to be endorsed by the EV Roaming Foundation, but since then, better ways to solve the same use cases have become available. The EV Roaming Foundation no longer endorses the document.

The lifecycle and its stages can be turned into a state diagram as follows:



2.3.2.1. Extension document lifecycle transitions

The EV Roaming Foundation's OCPI Development Working Group decides if and when an extension document transitions to a certain stage.

When deciding if an extension document can progress to the next lifecycle stage, the OCPI Development Working Group should consider the following general criteria:

- Whether the extension addresses business use cases that are encountered by parties in the market, and documents these business use cases.
- Whether the extension can be combined with other known standard extensions without conflicts or harmful interference of functionalities.
- Whether the extension is truly backward and forward compatible. This involves checking not only whether the extension is using only intended extension points documented in Extension points, but also whether receiving systems still assign the right meaning to messages they receive in case only one of the sender and the receiver supports the extension.
- Whether there is a sufficient audience for the extension. At least, there should be three parties committing to implement the extension. Ideally, there should be two separate connections, with four different parties that are committing to implementing the extension before it becomes accepted.
- No hostile manoeuvres. That is, the extension must not disadvantage an identifiable minority of OCPI users.

Specific criteria to check for on certain transitions:

- **proposal** to **rejected**: this should happen when there is no business use case that fulfills the above general criteria.
- **proposal** to **draft**: this should happen when an extension is proposed and there does seem to be a business use case fulfilling the above general criteria.
- **draft** to **under review**: this should happen when the EV Roaming Foundation Full Contributors working on the proposal have reached a stable draft that they feel is complete, but the proposal has not been reviewed by Associate Contributors or the general public yet. At this point the Development Working Group has to decide whether the proposal is to be reviewed by only Associate Contributors, or by the general public.
- **under review** to **rejected**: this should happen when only wider review by Associate Contributors and/or the general public reveals that the proposal does not fulfill the above general criteria. This is a hopefully rare unhappy transition, because if the proposal is so deficient, this should have become apparent in an earlier lifecycle stage.
- **under review** to **awaiting implementation**: this should happen when the comments from a wider review by Associate Contributors and/or the general public have been taken into account, and a stable draft has been reached after that. The expectation is that it will become Accepted or Incorporated, after perhaps minor adjustments to address difficulties encountered during implementation.
- **under review** to **accepted** or **incorporated**: this may happen when the criteria for a transition from "under review" to "awaiting implementation" are met, but in addition to that, there are serious reasons to not wait for implementation in practice. An example of such serious reasons could be regulatory requirements being imposed by a certain date.
- **awaiting implementation** to **accepted**: this should happen when the proposal has been implemented by a sufficient number of Platforms, and necessary adjustments discovered during implementation have been made to the proposal, but the extension is not considered by the Development Working Group to be so fundamental that incorporating it into the core specification is clearer than keeping it as a separate paper in "accepted" state.
- **awaiting implementation** to **incorporated**: this should happen when the proposal has been implemented by a sufficient number of Platforms, and necessary adjustments discovered during implementation have been made to the proposal, and the extension is considered by the Development Working Group to be so fundamental that incorporating it into the core specification is clearer than keeping it as a separate paper in "accepted" state.
- **awaiting implementation** to **rejected**: this should happen when only implementation work reveals that the proposal does not fulfill the above general criteria. This is a hopefully rare unhappy transition, because if the proposal is so deficient, this should have become apparent in an earlier lifecycle stage.
- **accepted** to **incorporated**: this should happen when an extension document that was previously in "accepted" state has become so widely adopted and relied upon that the Development Working Group believes that it is better for it to become incorporated in the core specification.
- **accepted** to **obsolete**: this should happen when an extension document that was previously in "accepted" state has, in the view of the Development Working Group, been superseded by newer ways of dealing with the business use case.

While the OCPI Development Working Group has the authority over lifecycle transitions, the day-to-day work on extension documents can be done in other venues like task groups, and these other venues can recommend the OCPI Development Working Group to transition the extension document to a new lifecycle stage.

2.3.2.2. Process tools and visibility

The EV Roaming Foundation will make available a repository with all extension documents that have ever been

submitted to the EV Roaming Foundation.

This repository will also contain a template document containing all the sections that an extension document must contain:

- A domain section with use cases that are addressed by the extension described in the document,
- A section on how the proposal deals with privacy concerns
- A section on how the proposal deals with security concerns
- A section on if and how the extension can be used with OCPI 2.3.0, and
- A section on if and how the extension can be used with OCPI 3.0.

2.3.3. Incompatible changes

There are scenarios in which it is the right course of action to make changes to OCPI that are not backward or forward compatible. Examples could be changing the transport from HTTP to another protocol, or changing the data format from JSON to something else.

In many cases, even changes at that level can be done in a backward and forward compatible way, by always starting the connection process in a way that conforms to the existing specification, and only deviating from the existing specification once both platforms have indicated support for the new extension to each other.

In some cases, such a bootstrapping process that conforms to the existing specification may not be possible or worthwhile. When that happens, an incompatible new version of OCPI can be developed. The process described in this paper does not cover that situation. When the OCPI Development Working Group decides that such an incompatible new version is to be developed, the process for doing so must be developed ad hoc, as it was for OCPI 2.x and for OCPI 3.0.

2.4. Extension points

As of OCPI version 2.3.0, OCPI has become extensible. By this, we mean that implementers are explicitly allowed by the specification to add new elements to OCPI messages, if they have a need that is not addressed by the existing specification. This section catalogues all extension points of OCPI 2.3.0 and 3.0. These extension points are the ways in which OCPI implementers can add elements to messages while remaining conformant to the OCPI specifications and interoperable with partner Platforms.

OCPI 3.0 offers implementers the following extension points:

- Open Enumerations
- JSON object fields
- Party-Issued Object Types
- RPC endpoints
- Types

OCPI 2.3.0 offers implementers the following extension points:

- Open Enumerations
- JSON object fields

- HTTP endpoints
- Types

When an extension uses only these extension points, the extension can be implemented while preserving compatibility with Platforms that do not implement the extension.

We will now describe every extension point in more detail.

2.4.1. Open enumerations

OCPI, since version 2.3.0, has different types for *open* and *closed* enumerations. The type for open enumerations is called OpenEnum; the type for closed enumerations is called enum.

Open enumerations are extension points where new values can be added to adapt to new realities. For example, when a new plug standard appears, a new value can be added to the ConnectorType OpenEnum.

2.4.2. JSON Object fields

OCPI uses the JSON ([JSON]) data format. JSON is designed for extensibility. Unlike other data exchange formats, JSON allows new fields to be added to message formats without requiring changes to systems that process messages of that format. That means that systems processing JSON messages are typically forward compatible, and new fields can be added easily in JSON-based message formats.

In the EV charging industry however, OCPP has tried to constrain this freedom that JSON provides by providing restrictive JSON Schema files for OCPP that consider OCPP messages invalid when they contain fields not foreseen in the OCPP specification.

With OCPI versions 2.3.0 and 3.0, OCPI is choosing a different direction. We believe that the forward compatibility that JSON encourages can be a great help in easier transitions to new protocol versions.

Adding new fields to JSON messages is a familiar and well-understood way to extend data formats for most developers nowadays. We therefore encourage people looking to extend OCPI to try to do it with just adding JSON fields before considering the more invasive extension points listed below.

2.4.3. Party-Issued Objects

OCPI, starting with version 3.0, specifies a generic mechanism for transferring data objects like Locations, Tariffs and CDRs. This generic mechanism is called the Party Issued Object system.

It is possible to extend OCPI by adding new object types that can be transferred using the Party Issued Object system. When adding a new Party Issued Object type, the party adding it must add requirements like those of the "Exchange information" use cases in the OCPI 3.0 specification. These requirements must specify:

- An object type definition for the objects to be transferred, and
- A string to use as the final path segment in party issued object update requests.

2.4.4. RPC endpoints

OCPI, starting with version 3.0, contains a generic mechanism for making Remote Procedure Calls (RPCs). The generic RPC mechanism is used when a party requests an operation from another party, like starting a charging session on an EVSE or authorizing a charge token for charging.

OCPI users can extend OCPI 3.0 by adding RPC endpoints for operations not previously supported by OCPI. To do so, they must provide requirements like those of the "Operations" use cases in the OCPI 3.0 specification. For each new RPC endpoint, the requirements must state:

- Whether the RPC is synchronous or asynchronous, and
- What the HTTP request verb is to be used (always POST for asynchronous RCPs)
- The operation name that is used in the endpoint URL path, and
- The object type of objects that can be sent in the requests to this endpoint, and
- The object type of objects that can be sent in the (synchronous or asynchronous) responses to this operation.

2.4.5. Types

It is possible to extend OCPI by adding entirely new data types, complementing those that are defined in the "Fundamental data type definitions" section of the OCPI Functional Use Cases document.

To do so, the extender must provide:

- A description of the possible values of the type, and
- A description of how values of the type are represented in JSON.

2.4.6. HTTP endpoints

In OCPI versions prior to OCPI 3.0, the Party Issued Object and RPC mechanisms did not exist. Equivalent extensions can be made by describing entirely new HTTP endpoints. When doing so, an extender must describe all properties of these endpoints, including but not necessarily limited to:

- What the URL to the endpoint is,
- What HTTP request verb should be used,
- What should be in the request payload,
- What should be in the response payload, and
- Which query parameters should be added to the URL.

2.5. Invariants

Invariants are the opposite of extension points: things that can absolutely not be changed about OCPI.

Real invariants in OCPI 3.0 are:

- OCPI messages are encoded as HTTP messages with bodies in the JSON format.
- OCPI messages are encrypted and authenticated with TLS.
- If a Party Issued Object changes, its version number changes as well.
- A Party Issued Object that was received from another Platform can only be changed by the same Party on the same Platform that initially sent it.

2.5.1. Pseudo-invariants

Pseudo-invariants are things that are true for many or all current implementations, but that are not in fact invariants of the OCPI protocol. Implementations may encounter OCPI messages or partner Platforms that do not adhere to these statements in the future.

Pseudo-invariants of OCPI 3.0 are:

- Only types defined in the base specification are used in OCPI messages.
- The root JSON value in OCPI messages is always an object.
- OCPI types are never union types.
- There is at most one Hub or other middlebox on the path between two Platforms communicating via a Hub.

2.6. Hubs and other "middleboxes"

"Middlebox" is a term used in IETF RFCs describing protocol evolution. Middleboxes are defined in [RFC3234] as any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host.

Roaming Hubs and other multi-party OCPI Platforms can be considered middleboxes in that they receive data via OCPI from one Party, and then forward that data via OCPI to another Party, typically while performing value-added functions like message routing, caching and enriching.

When Parties communicate through Hubs, a degree cooperation of the Hub is necessary for two Parties to use OCPI extensions in the communication among them. As such, Hubs that fail to support OCPI extensions can hold back the adoption of these extensions among Parties using OCPI.

Therefore, it is important that this paper puts requirements on Hubs to pass through fields and messages related to extensions, to make it unambiguous whether and how extensions can be used in OCPI communication that goes through one or more Hubs.

Roaming Hubs and other OCPI middleboxes have the following requirements to take into account when forwarding messages and data to and from OCPI connections that differ in their versions or their sets of supported extensions.

2.6.1. Requirements for middleboxes handling open enumerations

The principle here is that a middlebox may apply special logic for open enumeration values that it recognizes, but must pass on values that it does not recognize as-is. The following two requirements formalize that.

When a middlebox receives an OpenEnum value that it does not recognize, it MUST pass on the message with this value, and include the value unaltered in forwarded messages to all connected systems to which the data or message containing the value would be forwarded if the value was one that the middlebox recognizes.

When a middlebox receives an OpenEnum value that is not in the base specification but that the middlebox does recognize, it MAY transform the message in a way that makes the message more likely to be acted upon appropriately by the Platform to which the message is forwarded.

2.6.2. Requirements for middleboxes handling JSON object fields

OCPI and this document place no specific requirements on how Hubs and other middleboxes handle fields that are

not in the OCPI specification and that they do not recognize.

2.6.3. Requirements for middleboxes handling Party Issued Objects

OCPI and this document place no specific requirements on how Hubs and other middleboxes handle Party Issued Object types that are not in the OCPI specification and that they do not recognize.

2.6.4. Requirements for middleboxes handling RPC endpoints

Because of the security risks of forwarding arbitary request verbs to arbitrary URL paths, we recommend OCPI middleboxes to not forward unrecognized Remote Procedure Calls.

When an OCPI Platform receives an HTTP request with a combination of request verb and URL path that it does not recognize, even if the request fits the OCPI specification's requirements for Remote Procedure Calls, the Platform SHOULD reject these requests with the appropriate HTTP status codes in the 4xx range.

2.6.5. Requirements for middleboxes handling types

As a middlebox Platform knows the type for all the fields that it recognizes, the middlebox Platform can only encounter extended types that it does not recognize in JSON fields that it also does not recognize.

In such cases, OCPI and this document put no requirements on how the Hub or other middlebox handles the unrecognized datatype in the unrecognized field, as specified under Requirements for middleboxes handling JSON object fields.

2.6.6. Requirements for middleboxes handling HTTP endpoints

Middlebox Platforms SHOULD NOT forward requests to unrecognized endpoints. There are too many degrees of freedom for such requests. Forwarding them would be insecure.

3. Extension discovery

For some extensions, it will be useful for Platforms to announce in advance that they support those extensions. If an extension adds a new RPC endpoint, for example, a Platform may prefer to learn about the presence of that extension without having to probe for it by firing a request to the endpoint. This section describes how Platforms can announce the extensions they support to other Platforms, and how these other Platforms can recognize them.

3.1. Extension discovery in OCPI 3.0

To show how to detect whether an extension is supported by a Platform, or by a Party on a Platform, the extension document for that extension SHOULD contain a "How to detect support in OCPI 3.0" section.

For many extensions, support can probably simply be detected from the presence of certain fields or open enumeration values. These extensions do not require explicit announcement.

Fatures that require explicit announcement are to be announced differently dependending of whether they're supported across the whole Platform, or only for certain Parties on the Platform.

3.1.1. Discovering extensions supported across the whole Platform

If the extension needs announcement and is supported across all Parties on the Platform, support for the extension SHOULD be announced in an "extensions" field in the OCPIVersionURL, OcpiConnectionRequest and OcpiConnectionResponse objects sent by the Platform.

The value of this field should be a key-value map, represented as a JSON object. The keys are strings, and each key identifies a supported extension. The values in the map can give extra information required by the receiving Platform to use this extension effectively with the sending Platform. As an example, we could imagine that a future `OcpiConnectionRequest` with an "extensions" field could look like this:

```
{
  "version": "3.0",
  "request_timeout": 15000,
  "base_url": "https://example.com/ocpi",
  "extensions": {
    "send_charging_station_display_logo": {},
    "message_encryption": {
      "algorithm": "ECC_GCM_AES256",
      "public_key": "1rFIyJTkTWAHdjxoWnYdQg=="
    }
  }
}
```

This would indicate that the Platform supports a hypothetical extension to send a logo to be displayed on a charger's screen identified by `"send_charging_station_display_logo"`. It also indicates that the Platform supports a hypothetical extension for encrypting messages between Platforms at the OCPI layer, identified by `"message_encryption"`, and configured with a crypto algorithm and a public key.

If an extension requires explicit announcement at the Platform level, its extension document SHOULD specify in its "How to detect support in OCPI 3.0" section:

- what the key is that identifies the extension, and
- what possible values are under that key in the key-value map.

If the extension has to be announced but can be used without exchanging parameters between the Platforms, the value in the key-value map should be specified to be an empty object, that is `{}`.

3.1.2. Discovering extensions supported only for specific Parties

If the extension needs announcement and is supported only for certain Parties on the Platform, support for the extension SHOULD be announced in a "extensions" field in the Party object in the PlatformInformation object.

This field follows the same format, and has the same rules applying to it, as the "extensions" field described above for extensions supported across the whole Platform. The only difference is that if an extension is listed in the "extensions" field in a Party object, that indicates only support for the extension for that specific Party.

3.1.3. Versioning

From OCPI 3.0 onward, OCPI releases will have version identifiers with two numbers, separated by a dot ('.'). An increment of the first number means a change that breaks backward compatibility. An increment of the second number means that changes were made, but backward compatibility is preserved. That is, Platforms that support different versions that differ only in the second number can connect and interoperate despite interpreting the

messages they receive using different versions of the OCPI specification. Platforms that support only different versions that differ in their first number are not interoperable.

This approach is informed by, but not the same as, Semantic Version ([SEMVER]). Semantic Versioning is a specification for versioning software dependencies, and is not applicable to specification documents like OCPI.

3.2. Extensibility discovery and negotiation in OCPI 2.3.0

To show how to detect whether an extension is supported by a Platform, or by a Party on a Platform, the extension document for that extension SHOULD contain a "How to detect support in OCPI 2.3.0" section.

For many extensions, support can probably simply be detected from the presence of certain fields or open enumeration values. These extensions do not require explicit announcement.

Extensions that require explicit announcement are to be announced differently dependending of whether they're supported across the whole Platform, or only for certain Parties on the Platform.

3.2.1. Discovering and negotiating extensions supported across the whole Platform

If the extension needs announcement and is supported across all Parties on the Platform, support for the extension SHOULD be announced in an "extensions" field in the response body to the version details endpoint.

The value of this field should be a key-value map, represented as a JSON object. The keys are strings, and each key identifies a supported extension. The values in the map can give extra information required by the receiving Platform to use this extension effectively with the sending Platform. As an example, we could imagine that a future response to a version details endpoint with an "extensions" field could look like this:

```
{
  "version": "2.3.0",
  "endpoints": [...hidden for brevity...],
  "extensions": {
    "send_charging_station_display_logo": {},
    "message_encryption": {
      "algorithm": "ECC_GCM_AES256",
      "public_key": "1rFIyJTkTWAHdjxoWnYdQg=="
    }
  }
}
```

This would indicate that the Platform suports a hypothetical extension to send a logo to be displayed on a charger's screen identified by `"send_charging_station_display_logo"`. It also indicates that the Platform supports a hypothetical extension for encrypting messages between Platforms at the OCPI layer, identified by `"message_encryption"`, and configured with a crypto algorithm and a public key.

If an extension requires explicit announcement at the Platform level, its extension document SHOULD specify in its "How to detect support in OCPI 2.3.0" section:

- what the key is that identifies the extension, and
- what possible values are under that key in the key-value map.

If the extension has to be announced but can be used without exchanging parameters between the Platforms, the value in the key-value map should be specified to be an empty object, that is {}.

3.2.2. Discovering and negotiating extensions supported only for specific Parties

If the extension needs announcement and is supported only for certain Parties on the Platform, support for the extension SHOULD be announced in an "extensions" field in the CredentialsRole object in the Credentials object.

This field follows the same format, and has the same rules applying to it, as the "extensions" field described above for extensions supported across the whole Platform. The only difference is that if an extension is listed in the "extensions" field in a CredentialsRole object, that indicates only support for the extension for that specific Party in that specific role.

3.2.3. Versioning

In the OCPI 2.x version family, versioning has not been consistent. Platforms SHOULD NOT assume that there is any backward compatibility between different versions in the 2.x family.

4. Actions to be taken by the OCPI editors to foster the extension development process

- Establish a cadence of yearly backward compatible releases, with a release cutoff deadline at a fixed date of the year. All extensions that the OCPI Development Working Group has transitioned to "accepted" and deemed fit for incorporation by the deadline date shall be included in that year’s scheduled release.
- Structure the OCPI Development Working Group around the different extensions that are worked on. In the beginning of the call, an overview of the extensions under development, their lifecycle stages and any imminent or recent lifecycle transitions should be provided.
- Provide an overview of all past and present extension proposals and their lifecycle stages online.
- Start hosting the OCPI Development Working Group under the extension development process described in this paper from Autumn 2026 onward.

5. Actions to be recommended by the OCPI Development Working Group to the EV Roaming Foundation Board to foster the extension development process

- Amend the Terms and Procedures to encourage Contributors to contribute their custom OCPI extensions as proposed standard extensions to the OCPI development process.
- Amend the Terms and Procedures to make it clear that to all material that a Contributor supplies under the extension document process, the same intellectual property rules apply as to material offered in the OCPI Development Working Group call. The goal here is that the OCPI specification and all extension documents remain available to read and implement free of charge.

6. Actions to be recommended by the OCPI Development Working Group to the EV Roaming Foundation Advisory Board to foster the extension document process

- Provide a clearer definition and operationalization of consensus, as it applies to decision-making by the OCPI Development Working Group.

7. References

Abbreviation	Title	URL
JSON	The JSON data interchange syntax	https://ecma-international.org/publications-and-standards/standards/ecma-404/
RFC3234	Middleboxes: Taxonomy and Issues	https://datatracker.ietf.org/doc/html/rfc3234
RFC6709	Design Considerations for Extensions	https://datatracker.ietf.org/doc/html/rfc6709
SEMVER	Semantic Versioning 2.0.0	https://semver.org/spec/v2.0.0.html