OCPP 2.0

Part 4 - Errata

v1.1, 2020-03-31

1. Disclaimer

Version History

Version	Date	Author	Description
1.1	2020-03-31	Franc Buve (*OCA*) Milan Jansen (*OCA*)	Version 1.1
1.0	2019-12-20	Franc Buve (*OCA*) Milan Jansen (*OCA*)	Version 1.0

2. Scope

This document contains errata on "part 4: JSON over WebSockets implementation guide" of the OCPP 2.0 documentation.

All these errata have been merged in the documentation, which has resulted in the release of OCPP 2.0.1.

2.1. Terminology and Conventions

Bold: when needed to clarify differences, bold text might be used.

3. Errata

3.1. Page 4, Section 3.1.1. The connection URL, The colon ":" character must not be used for the identity of a Charging Station

The specification is describing the identifierString as the dataType for the Charging Station identity. This is correct, but there is one exception. The colon ":" character might not be used, because the unique identifier is also used for the basic authentication username. The colon ":" character is used to separate the basic authentication username and the password.

Old text	The Charging Station identity datatype is identifierString (For definition see [OCPP2.0-PART2] The maximum length of the Charging Station identity is: 48 (Note: Maximum length was chosen to ensure compatibility with EVSE ID from [EMI3] "Part 2: business objects."
New text	The Charging Station identity datatype is identifierString (For definition see [OCPP2.0-PART2] **Additionally the colon ":" character might not be used, because the unique identifier is also used for the basic authentication username. The colon ":" character is used to separate the basic authentication username and the password.** The maximum length of the Charging Station identity is: 48 (Note: Maximum length was chosen to ensure compatibility with EVSE ID from [EMI3] "Part 2: business objects."

3.2. Page 6, Section 3.3. WebSocket Compression, Description WebSocket Compression optional for Charging Stations not clear

Description about WebSocket Compression handshake in RFC 7692 is causing a bit of confusion. It is not clear if compression is required or not.

Old text	RFC 7692 allows the Charging Station and the CSMS to do a negotiation during the connection setup. When both parties support the Compression Extension they will then use DEFLATE compression ([RFC1951]) when sending data over the line. When one of the parties doesn't support it, the JSON will be send uncompressed (like in OCPP 1.6J).
New text	**OCPP Requires the CSMS (and Local Controller) to support RFC 7692, WebSocket compression is seen as a relative simple way to reduce mobile data usage. For a Charging Station this is not a hard requirement, as this might be more complex to implement on an embedded platform, but as this is seen as efficient solution to reduce mobile data usage, it is RECOMMENDED to be implemented on a Charging Station that uses a mobile data connection.** RFC 769 allows the Charging Station and the CSMS to do a negotiation during the connection setup. When both parties support the Compression Extension they will then use DEFLATE compression ([RFC1951]) when sending data over the line. When one of the parties doesn't support it, the JSON will be sent uncompressed (like in OCPP 1.6J). **When the Charging Station detects that compression is not used, it is RECOMMENDED not to close the connection, as turning of compression can be very useful during development, testing and debugging.**

3.3. Page 6, Section 4.1.1. Synchronicity, Unclear when the CSMS is allowed to send messages

It is unclear that the CSMS does not have to wait for a response from Charging Station 1 when wanting to send a request to Charging Station 2.

Old text	A Charging Station or CSMS SHALL NOT send a CALL message to the other party unless all the CALL messages it sent before have been responded to or have timed out. A CALL message has been responded to when a CALLERROR or CALLRESULT message has been received with the message ID of the CALL message.
New text	A Charging Station or CSMS SHALL NOT send a CALL message to the other party unless all the CALL messages it sent before have been responded to or have timed out. **This does not mean that the CSMS cannot send a message to another Charging Station, while waiting for a response of a first Charging Station, this rule is per OCPP-J connection.** A CALL message has been responded to when a CALLERROR or CALLRESULT message has been received with the message ID of the CALL message.

3.4. Page 10, Section 4.2.3, Unclear if a system is allowed to drop a message when it is not conform the JSON schema

There are error codes like; PropertyConstraintViolation, OccurrenceConstraintViolation and TypeConstraintViolation. But there is no remark or requirement that explicitly allows a system to delete a complete message when it is not conform the JSON schema.

```
When a message contains any invalid OCPP and/or it is not conform the JSON
schema, the system is allowed to drop the message.
```

3.5. Page 10, Section 4.2.3, Missing CALLERROR example

The following example plus description needs be added, below the CALLERROR field descriptions.

For example, a CALLERROR could look like this:

```
[4,
  "162376037",
  "NotSupported",
  "SetDisplayMessageRequest not implemented",
  {}
]
```

3.6. Page 15, Section 7.2. Handling Signed Messages, incomplete requirement

The requirement is also applicable for the CSMS, not only the Charging Station.

Old text	When a Charging Station receives a signed request, and it supports digital signing, it SHALL send a signed reply.
New text	When a Charging Station **or CSMS** receives a signed request, and it supports digital signing, it SHALL send a signed reply.

3.7. Page 17, Section 8.4. WebSocketPingInterval, A recommendation needs to be added about how to configure the WebSocketPingInterval

Changed description:

Old description	A value of 0 disables client side websocket Ping / Pong. In this case there is either no ping / pong or the server initiates the ping and client responds with Pong. Positive values are interpreted as number of seconds between pings. Negative values are not allowed, SetConfiguration is then expected to return a Rejected result.
New description	A value of 0 disables client side websocket Ping / Pong. In this case there is either no ping / pong or the server initiates the ping and client responds with Pong. Positive values are interpreted as number of seconds between pings. Negative values are not allowed, SetConfiguration is then expected to return a Rejected result. **It is recommended to configure WebSocketPingInterval smaller then: MessageAttemptsTransactionEvent * MessageAttemptIntervalTransactionEvent. This will limit the chance of the resend mechanism for transaction-related messages being triggered by connectivity issues.**

3.8. Page 18, A section needs to be added that explains how to use the CustomData Extension

In the JSON schema files all classes have the attribute *additionalProperties* set to *false*, such that a JSON parser will not accept any other properties in the message. In order to allow for some flexibility to create non-standard extensions for experimentation purposes, every JSON class has been extended with a "customData" property. This property is of type "CustomDataType", which has only one required property: "vendorId", which is used to identify the kind of customization. However, since it does not have *additionalProperties* set to *false* it can be freely extended with new properties.

In the same way as is defined for the DataTransfer message, the "vendorId" should be a value from the reversed DNS namespace, where the top tiers of the name, when reversed, should correspond to the publicly registered primary DNS name of the Vendor organization.

The following example shows the "CustomDataType" definition and the (optional) "customData" property in the schema definition of HeartbeatRequest:

{ "$schema": "http://json-schema.org/draft-06/schema#", "$id": "HeartbeatRequest", "definitions": { "CustomDataType": { "description": "This class does not get 'AdditionalProperties = false' in the schema generation, so it can be extended with arbitrary JSON properties to allow adding custom data.", "javaType": "CustomData", "type": "object", "properties": { "vendorId": { "type": "string", "maxLength": 255 } }, "required": ["vendorId"] } }, "type": "object", "additionalProperties": false, "properties": { "customData": { "$ref": "#/definitions/CustomDataType" } } }

Whereas the standard HeartbeatRequest has an empty body, a customized version, that provides the value of the main meter and a count of all sessions to date, could look like this:

{ "customData": { "vendorId": "com.mycompany.customheartbeat", "mainMeterValue": 12345, "sessionsToDate": 342 } }

A CSMS that has implemented this extension, identified by its "vendorId", will be able to process the data. Other CSMS implementations will simply ignore these custom properties.

A CSMS can request a report of the *CustomizationCtrlr* component to get a list of all customizations that are supported by the Charging Station.