Open Charge Point Protocol 1.6

edition 2 FINAL, 2017-09-28

Table of Contents

Interface description between Charge Point and Central System

Document Version	1.6 edition 2
Document Status	FINAL
Document Release Date	2017-09-28

Version History

VERSION	DATE	AUTHOR	DESCRIPTION
1.6 edition 2	2017-09-28	Robert de Leeuw *IHomer* Brendan McMahon *ESB ecars* Klaas van Zuuren *ElaadNL*	OCPP 1.6 edition 2 Final release. Contains all of the known erratas (including v3.0) and improved styling.
1.6	2015-10-08	Robert de Leeuw *IHomer* Reinier Lamers *The New Motion* Brendan McMahon *ESB ecars* Lambert Muhlenberg *Alfen* Patrick Rademakers *IHomer* Sergiu Tcaciuc *smartlab* Klaas van Zuuren *ElaadNL*	1.6 Final Release. For changes relative to 1.5, see appendix New in OCPP 1.6.
1.5	2012-06-01	Franc Buve	Specification ready for release. Includes: CR-01 Authentication/authorization lists CR-02 Interval meter readings CR-03 Charge point reservation CR-04 Generic data transfer CR-05 More detailed status notifications CR-06 Query configuration parameters CR-07 Timestamp in BootNotification mandatory CR-08 Response to StartTransaction.req with status other than Accepted is not clearly defined CR-09 Increase size of firmwareVersion in BootNotification
1.2	2011-02-21	Franc Buve	
1.0	2010-10-19	Franc Buve	Final version approved by e-laad.nl. Identical to version 0.12.

Chapter 1. Scope

This document defines the protocol used between a **Charge Point** and **Central System**. If the protocol requires a certain action or response from one side or the other, then this will be stated in this document.

The specification does not define the communication technology. Any technology will do, as long as it supports TCP/IP connectivity.

Chapter 2. Terminology and Conventions

2.1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119], subject to the following additional clarification clause:

The phrase "valid reasons in particular circumstances" relating to the usage of the terms "SHOULD", "SHOULD NOT", "RECOMMENDED", and "NOT RECOMMENDED" is to be taken to mean technically valid reasons, such as the absence of necessary hardware to support a function from a charge point design: for the purposes of this specification it specifically excludes decisions made on commercial, or other non-technical grounds, such as cost of implementation, or likelihood of use.

All sections and appendixes, except "Scope" and "Terminology and Conventions", are normative, unless they are explicitly indicated to be informative.

2.2. Definitions

This section contains the terminology that is used throughout this document.

Central System	Charge Point Management System: the central system that manages Charge Points and has the information for authorizing users for using its Charge Points.
CiString	Case Insensitive String. Only printable ASCII allowed.
Charge Point	The Charge Point is the physical system where an electric vehicle can be charged. A Charge Point has one or more connectors.
Charging Profile	Generic Charging Profile, used for different types of Profiles. Contains information about the Profile and holds the Charging Schedule. In future versions of OCPP it might hold more than 1 Charging Schedule.
Charging Schedule	Part of a Charging Profile. Defines a block of charging Power or Current limits. Can contain a start time and length.
Charging Session	A Charging Session is started when first interaction with user or EV occurs. This can be a card swipe, remote start of transaction, connection of cable and/or EV, parking bay occupancy detector, etc.
Composite Charging Schedule	The charging schedule as calculated by the Charge Point. It is the result of the calculation of all active schedules and possible local limits present in the Charge Point. Local Limits might be taken into account.
Connector	The term "Connector", as used in this specification, refers to an independently operated and managed electrical outlet on a Charge Point. This usually corresponds to a single physical connector, but in some cases a single outlet may have multiple physical socket types and/or tethered cable/connector arrangements to facilitate different vehicle types (e.g. four-wheeled EVs and electric scooters).
Control Pilot signal	Signal used by a Charge Point to inform EV of maximum Charging power or current limit, as defined by [IEC61851-1].
Energy Offer Period	Energy Offer Period starts when the EVSE is ready and willing to supply energy.
Energy Offer SuspendPeriod	During a transaction, there may be periods the EnergyOffer to EV is suspended by the EVSE, for instance due to Smart Charging or local balancing.
Energy Transfer Period	Time during which an EV chooses to take offered energy, or return it. Multiple Energy Transfer Periods are possible during a Transaction.

Local Controller	Optional device in a smart charging infrastructure. Located on the premises with a number of Charge Points connected to it. Sits between the Charge Points and Central System. Understands and speaks OCPP messages. Controls the Power or Current in other Charge Point by using OCPP smart charging messages. Can be a Charge Point itself.
OCPP-J	OCPP via JSON over WebSocket
OCPP-S	OCPP via SOAP
Phase Rotation	Defines the wiring order of the phases between the electrical meter (or if absent, the grid connection), and the Charge Point connector.
Transaction	The part of the charging process that starts when all relevant preconditions (e.g. authorization, plug inserted) are met, and ends at the moment when the Charge Point irrevocably leaves this state.
String	Case Sensitive String. Only printable ASCII allowed. All strings in messages and enumerations are case sensitive, unless explicitly stated otherwise.

2.3. Abbreviations

CSL	Comma Separated List
CPO	Charge Point Operator
DNS	Domain Name System
DST	Daylight Saving Time
EV	Electrical Vehicle, this can be BEV (battery EV) or PHEV (plug-in hybrid EV)
EVSE	Electric Vehicle Supply Equipment [IEC61851-1]
FTP (S)	File Transport Protocol (Secure)
HTTP (S)	HyperText Transport Protocol (Secure)
ICCID	Integrated Circuit Card Identifier
IMSI	International Mobile Subscription Identity
JSON	JavaScript Object Notation
NAT	Native Address Translation
PDU	Protocol Data Unit
SC	Smart Charging
SOAP	Simple Object Access Protocol
URL	Uniform Resource Locator
RST	3 phase power connection, Standard Reference Phasing

RTS	3 phase power connection, Reversed Reference Phasing
SRT	3 phase power connection, Reversed 240 degree rotation
STR	3 phase power connection, Standard 120 degree rotation
TRS	3 phase power connection, Standard 240 degree rotation
TSR	3 phase power connection, Reversed 120 degree rotation
UTC	Coordinated Universal Time

2.4. References

[IEC61851-1]	"IEC 61851-1 2010: Electric vehicle conductive charging system - Part 1: General requirements" https://webstore.iec.ch/publication/
[OCPP1.5]	"OCPP 1.5: Open Charge Proint Protocol 1.5" http://www.openchargealliance.org/downloads/
[OCPP_1.6CT]	"OCPP 1.6 Compliance testing" http://www.openchargealliance.org/downloads/
[OCPP_IMP_J]	"OCPP JSON Specification" http://www.openchargealliance.org/downloads/
[OCPP_IMP_S]	"OCPP SOAP Specification" http://www.openchargealliance.org/downloads/
[RFC2119]	"Key words for use in RFCs to Indicate Requirement Levels". S. Bradner. March 1997. http://www.ietf.org/rfc/rfc2119.txt

Chapter 3. Introduction

This is the specification for OCPP version 1.6.

OCPP is a standard open protocol for communication between Charge Points and a Central System and is designed to accommodate any type of charging technique.

OCPP 1.6 introduces new features to accommodate the market: Smart Charging, OCPP using JSON over Websockets, better diagnostics possibilities (Reason), more Charge Point Statuses and TriggerMessage. OCPP 1.6 is based on OCPP 1.5, with some new features and a lot of textual improvements, clarifications and fixes for all known ambiguities. Due to improvements and new features, OCPP 1.6 is not backward compatible with OCPP 1.5.

For a full list of changes, see: New in OCPP 1.6.

Some basic concepts are explained in the sections below in this introductory chapter. The chapters: Operations Initiated by Charge Point and Operations Initiated by Central System describe the operations supported by the protocol. The exact messages and their parameters are detailed in the chapter: Messages and data types are described in chapter: Types. Defined configuration keys are described in the chapter: Standard Configuration Key Names & Values.

3.1. Edition 2

This document is OCPP 1.6 edition 2. This document still describes the same protocol: OCPP 1.6, only the documentation is improved. On message level there are no changes compared to the original release of OCPP 1.6 of October 2015. All known errata (previously published in a separate document) have been merged into this document, making it easier for the implementers to work with the specification. When there is doubt about the way OCPP 1.6 should be implemented, this document over rules the original document.

3.2. Document structure

With the introduction of OCPP 1.6, there are two different flavours of OCPP; next to the SOAP based implementations, there is the possibility to use the much more compact JSON alternative. To avoid confusion in communication on the type of implementation we recommend using the distinct suffixes -J and -S to indicate JSON or SOAP. In generic terms this would be OCPP-J for JSON and OCPP-S for SOAP.

To support the different flavours, the OCPP standard is divided in multiple documents. The base document (the one you are reading now) contains the technical protocol specification. The technical protocol specification must be used with one of the transport protocol specifications. the OCPP SOAP Specification contains the implementation specification needed to make a OCPP-S implementation. For OCPP-J, the OCPP JSON Specification must be used.

For improved interoperabillity between the Central Systems and Charge Points, it is adviced to meet the requirements stated in the OCPP 1.6 Compliance testing documentation.

3.3. Feature Profiles

This section is normative.

In OCPP 1.6 features and associated messages are grouped in *profiles*. Depending on the required functionality, implementers can choose to implement one or more of the following profiles.

PROFILE NAME	DESCRIPTION
Core	Basic Charge Point functionality comparable with OCPP 1.5 [OCPP1.5] without support for firmware updates, local authorization list management and reservations.
Firmware Management	Support for firmware update management and diagnostic log file download.

PROFILE NAME	DESCRIPTION
Local Auth List Management	Features to manage the local authorization list in Charge Points.
Reservation	Support for reservation of a Charge Point.
Smart Charging	Support for basic Smart Charging, for instance using control pilot.
Remote Trigger	Support for remote triggering of Charge Point initiated messages

These profiles can be used by a customer to determine if a OCPP 1.6 product has the required functionality for their business case. Compliance testing will test per profile if a product is compliant with the OCPP 1.6 specification.

Implementation of the Core profile is required. Other profiles are optional.

When the profiles **Core** , **Firmware Management** , **Local Auth List Management** and **Reservation** are implemented, all functions originating from OCPP 1.5 [OCPP1.5] are covered.

The grouping of all messages in their profiles can be found in the table below.

MESSAGE	CORE	FIRMWARE MANAGEMENT	LOCAL AUTH LIST MANAGEMENT	REMOTE TRIGGER	RESERVATION	SMART CHARGING
Authorize	X					
BootNotification	X					
ChangeAvailability	X					
ChangeConfiguration	X					
ClearCache	X					
DataTransfer	X					
GetConfiguration	X					
Heartbeat	X					
MeterValues	X					
RemoteStartTransaction	X					
RemoteStopTransaction	X					
Reset	X					
StartTransaction	X					
StatusNotification	X					
StopTransaction	X					

MESSAGE	CORE	FIRMWARE MANAGEMENT	LOCAL AUTH LIST MANAGEMENT	REMOTE TRIGGER	RESERVATION	SMART CHARGING
UnlockConnector	X					
GetDiagnostics		X				
DiagnosticsStatusNotification		X				
FirmwareStatusNotification		X				
UpdateFirmware		X				
GetLocalListVersion			X			
SendLocalList			X			
CancelReservation				X		
ReserveNow				X		
ClearChargingProfile					X	
GetCompositeSchedule					X	
SetChargingProfile					X	
TriggerMessage						X

The support for the specific feature profiles is reported by the `SupportedFeatureProfiles` configuration key.

3.4. General views of operation

This section is informative.

The following figures describe the general views of the operations between Charge Point and Central System for two cases:

1. a Charge Point requesting authentication of a card and sending charge transaction status
2. Central System requesting a Charge Point to update its firmware

The arrow labels in the following figures indicate the PDUs exchanged during the invocations of the operations. These PDUs are defined in detail in the Messages section.

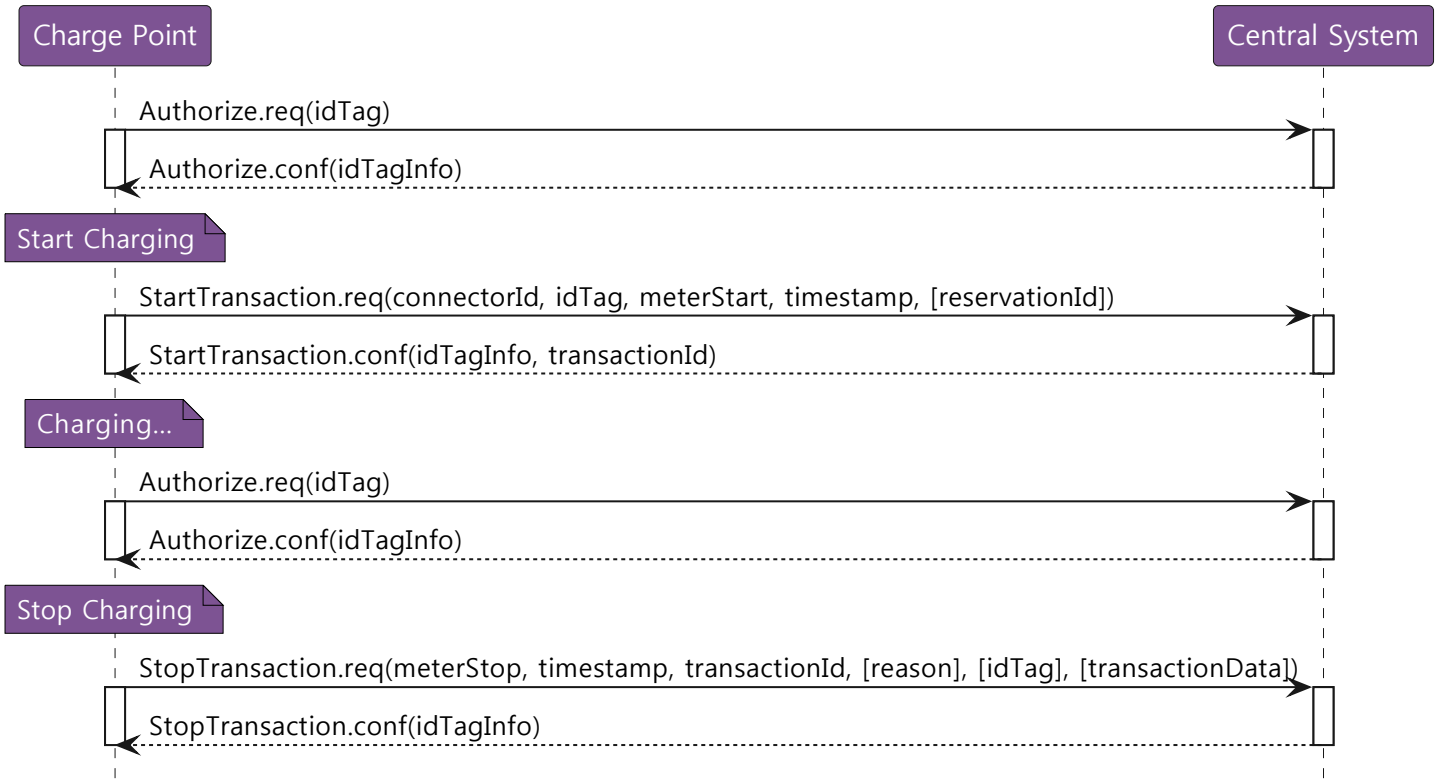

Figure 1. Sequence Diagram: Example of starting and stopping a transaction

When a Charge Point needs to charge an electric vehicle, it needs to authenticate the user first before the charging can be started. If the user is authorized the Charge Point informs the Central System that it has started with charging.

When a user wishes to unplug the electric vehicle from the Charge Point, the Charge Point needs to verify that the user is either the one that initiated the charging or that the user is in the same group and thus allowed to terminate the charging. Once authorized, the Charge Point informs the Central System that the charging has been stopped.

A Charge Point MUST NOT send an Authorize.req before stopping a transaction if the presented idTag is the same as the idTag presented to start the transaction.

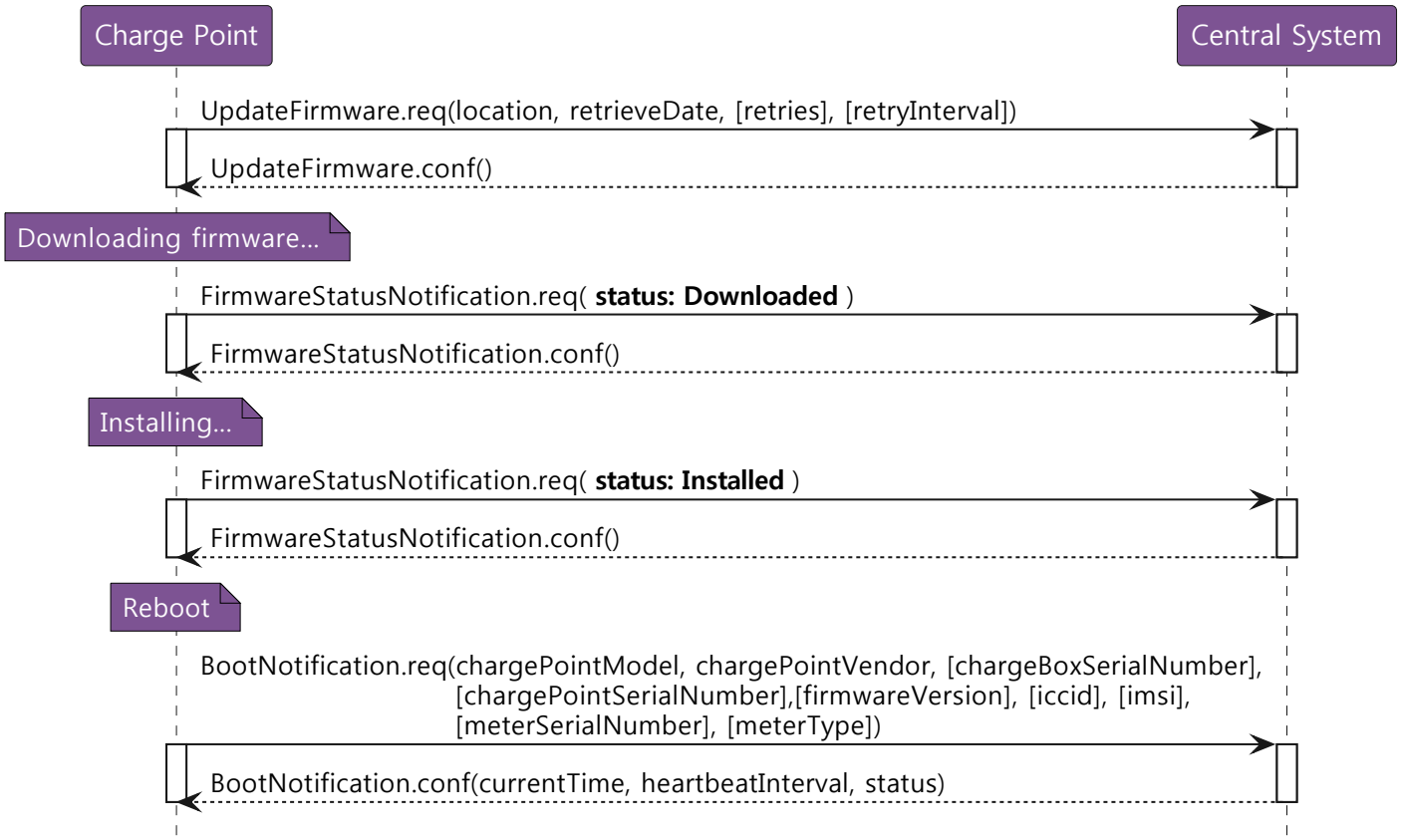

Figure 2. Sequence Diagram: Example of a firmware update

When a Charge Point needs to be updated with new firmware, the Central System informs the Charge Point of the time at which the Charge Point can start downloading the new firmware. The Charge Point SHALL notify the Central System after each step as it downloads and installs the new firmware.

3.5. Local Authorization & Offline Behavior

This section is normative.

In the event of unavailability of the communications or even of the Central System, the Charge Point is designed to operate stand-alone. In that situation, the Charge Point is said to be *offline*.

To improve the experience for users, a Charge Point MAY support local authorization of identifiers, using an Authorization Cache and/or a Local Authorization List.

This allows (a) authorization of a user when *offline* , and (b) faster (apparent) authorization response time when communication between Charge Point and Central System is slow.

The `LocalAuthorizeOffline` configuration key controls whether a Charge Point will authorize a user when *offline* using the Authorization Cache and/or the Local Authorization List.

The `LocalPreAuthorize` configuration key controls whether a Charge Point will use the Authorization Cache and/or the Local Authorization List to start a transaction without waiting for an authorization response from the Central System.

A Charge Point MAY support the (automatic) authorization of any presented identifier when *offline* , to avoid refusal of charging to bona-fide users that cannot be explicitly authorized by Local Authorization List/Authorization Cache entries. This functionality is explained in more detail in Unknown Offline Authorization.

3.5.1. Authorization Cache

A Charge Point MAY implement an *Authorization Cache* that autonomously maintains a record of previously presented identifiers that have been successfully authorized by the Central System. (*Successfully* meaning: a response received on a message containing an

idTag)

If implemented, the Authorization Cache SHOULD conform to the following semantics:

- The Cache contains all the latest received identifiers (i.e. valid and NOT-valid).
- The Cache is updated using all received IdTagInfo (from Authorize.conf, StartTransaction.conf and StopTransaction.conf)
- When the validity of a Cache entry expires, it SHALL be changed to expired in the Cache.
- When an IdTagInfo is received for an identifier in the Cache, it SHALL be updated.
- If new identifier authorization data is received and the Authorization Cache is full, the Charge Point SHALL remove any NOT-valid entries, and then, if necessary, the oldest valid entries to make space for the new entry.
- Cache values SHOULD be stored in non-volatile memory, and SHOULD be persisted across reboots and power outages.
- When an identifier is presented that is stored in the cache as NOT-valid, and the Charge Point is *online* : an Authorize.req SHOULD be sent to the central System to check the current state of the identifier.

Operation of the Authorization Cache, when present, is reported (and controlled, where possible) by the `AuthorizationCacheEnabled` configuration key.

3.5.2. Local Authorization List

The Local Authorization List is a list of identifiers that can be synchronized with the Central System.

The list contains the authorization status of all (or a selection of) identifiers and the authorization status/expiration date.

Identifiers in the Local Authorization list can be marked as **valid** , **expired** , **(temporarily) blocked** , or **blacklisted** , corresponding to IdTagInfo *status* values *Accepted* / *ConcurrentTx* , *Expired* , *Blocked* , and *Invalid*, respectively.

These values may be used to provide more fine grained information to users (e.g. by display message) during local authorization.

The Local Authorization List SHOULD be maintained by the Charge Point in non-volatile memory, and SHOULD be persisted across reboots and power outages.

A Charge Point that supports Local Authorization List SHOULD implement the configuration key: LocalAuthListMaxLength This gives the Central System a way to known the the maximum possible number of Local Authorization List elements in a Charge Point

The Charge Point indicates whether the Local Authorization List is supported by the presence or absence of the `LocalAuthListManagement` element in the value of the `SupportedFeatureProfiles` configuration key.

Whether the Local Authorization List is enabled is reported and controlled by the LocalAuthListEnabled configuration key.

The Central System can synchronize this list by either (1) sending a complete list of identifiers to replace the Local Authorization List or (2) by sending a list of changes (add, update, delete) to apply to the Local Authorization List. The operations to support this are Get Local List Version and Send Local List.

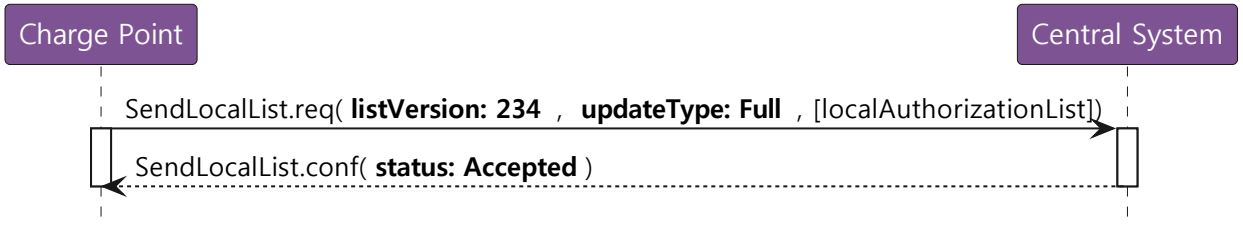

Figure 3. Sequence Diagram: Example of a full local authorization list update

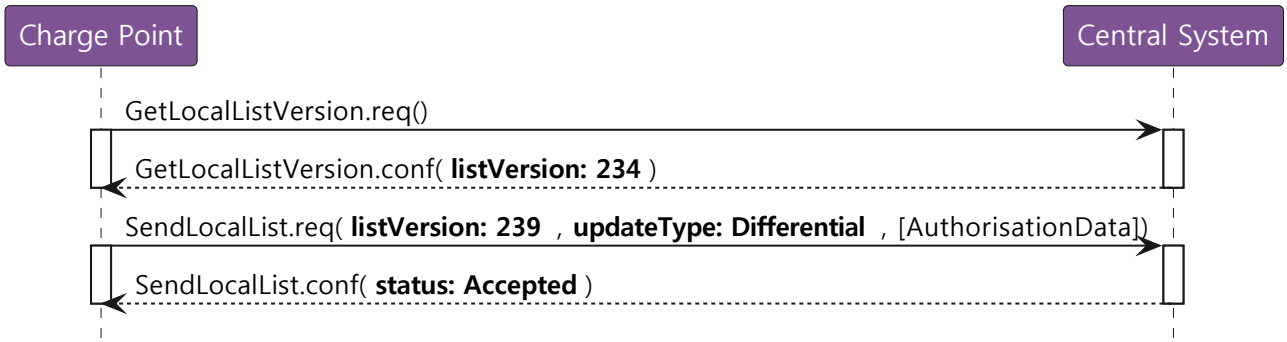

Figure 4. Sequence Diagram: Example of a differential local authorization list update

The Charge Point SHALL NOT modify the contents of the Authorization List by any other means than upon a the receipt of a SendLocalList PDU from the Central System.

Conflicts between the local authorization list and the validity reported in, for instance, a StartTransaction.conf message might occur. When this happens the Charge Point SHALL inform the Central System by sending a StatusNotification with ConnectorId set to 0, and ErrorCode set to 'LocalListConflict'.

3.5.3. Relation between Authorization Cache and Local Authorization List

The Authorization Cache and Local Authorization List are distinct logical data structures. Identifiers known in the Local Authorization List SHALL NOT be added to the Authorization Cache.

Where both Authorization Cache and Local Authorization List are supported, a Charge Point SHALL treat Local Authorization List entries as having priority over Authorization Cache entries for the same identifiers.

3.5.4. Unknown Offline Authorization

When *offline*, a Charge Point MAY allow automatic authorization of any "unknown" identifiers that cannot be explicitly authorized by Local Authorization List or Authorization Cache entries. Identifiers that are present in a Local Authorization List that have a status other than "Accepted" (Invalid, Blocked, Expired) MUST be rejected.
Identifiers that were valid but are apparently expired due to passage of time MUST also be rejected

Operation of the Unknown Offline Authorization capability, when supported, is reported (and controlled, where possible) by the `AllowOfflineTxForUnknownId` configuration key.

When connection to the Central Server is restored, the Charge Point SHALL send a Start Transaction request for any transaction that was authorized offline , as required by transaction-related message handling. When the authorization status in the StartTransaction.conf is not *Accepted*, and the transaction is still ongoing, the Charge Point SHOULD:

- when `StopTransactionOnInvalidId` is set to *true*: stop the transaction normally as stated in Stop Transaction. The Reason field in the Stop Transaction request should be set to DeAuthorized. If the Charge Point has the possibility to lock the Charging Cable, it SHOULD keep the Charging Cable locked until the owner presents his identifier.
- when StopTransactionOnInvalidId is set to *false*: only stop energy delivery to the vehicle.

In the case of an invalid identifier, an operator MAY choose to charge the EV with a minimum amount of energy so the EV is able to drive away. This amount is controlled by the optional configuration key: MaxEnergyOnInvalidId.

3.6. Transaction in relation to Energy Transfer Period

This section is informative.

The Energy Transfer Period is a period of time during wich energy is transferred between the EV and the EVSE.
There MAY be multiple Energy Transfer Periods during a Transaction

Multiple Energy Transfer Periods can be separated by either:

- an EVSE-initiated supense of transfer during which the EVSE does not offer energy transfer
- an EV-initiated suspense of transfer during which the EV remains electrically connected to the EVSE
- an EV-initiated suspense of transfer during which the EV is not electrically connected to the EVSE.

A Central System MAY deduce the start and end of an Energy Transfer Period from: the MeterValues that are sent during the Transaction, the status notifications: Charging, SuspendedEV and/or SuspendedEVSE. etc.
Central System implementations need to take into account factors such as: Some EVs don't go to state SuspendedEV: they might continue to trickle charge. Some Charge Point don't even have a electrical meter.

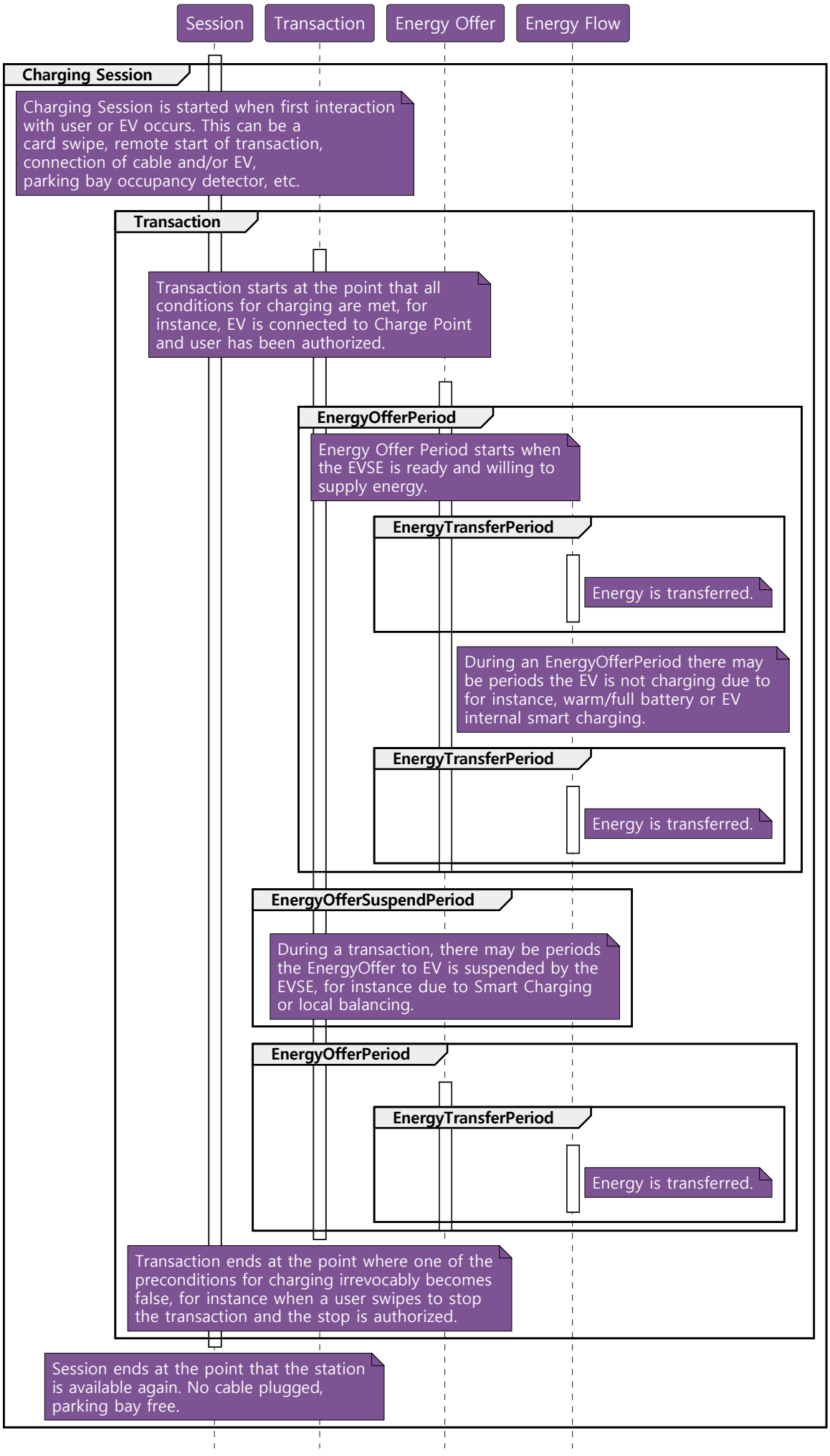

Figure 5. OCPP Charging Session and transaction definition

3.7. Transaction-related messages

This section is normative.

The Charge Point SHOULD deliver transaction-related messages to the Central System in chronological order as soon as possible. Transaction-related messages are StartTransaction.req, StopTransaction.req and periodic or clock-aligned MeterValues.req messages.

When *offline* , the Charge Point MUST queue any transaction-related messages that it would have sent to the Central System if the Charge Point had been online.

In the event that a Charge Point has transaction-related messages queued to be sent to the Central System, new messages that are not transaction-related MAY be delivered immediately without waiting for the queue to be emptied. It is therefore allowed to send, for example, an Authorize request or a Notifications request before the transaction-related message queue has been emptied, so that customers are not kept waiting and urgent notifications are not delayed.

The delivery of new transaction-related messages SHALL wait until the queue has been emptied. This is to ensure that transaction-related messages are always delivered in chronological order.

When the Central System receives a transaction-related message that was queued on the Charge Point for some time, the Central System will not be aware that this is a historical message, other than by inference given that the various timestamps are significantly in the past. It SHOULD process such a message as any other.

3.7.1. Error responses to transaction-related messages

It is permissible for the Charge Point to skip a transaction-related message if and only if the Central System repeatedly reports a 'failure to process the message'. Such a stipulation is necessary, because otherwise the requirement to deliver every transaction-related message in chronological order would entail that the Charge Point cannot deliver any transaction-related messages to the Central System after a software bug causes the Central System not to acknowledge one of the Charge Point's transaction-related messages.

What kind of response, or failure to respond, constitutes a 'failure to process the message' is defined in the documents OCPP JSON Specification and OCPP SOAP Specification.

The number of times and the interval with which the Charge Point should retry such failed transaction-related messages MAY be configured using the TransactionMessageAttempts and TransactionMessageRetryInterval configuration keys.

When the Charge Point encounters a first failure to deliver a certain transaction-related message, it SHOULD send this message again as long as it keeps resulting in a failure to process the message and it has not yet encountered as many failures to process the message for this message as specified in its TransactionMessageAttempts configuration key. Before every retransmission, it SHOULD wait as many seconds as specified in its TransactionMessageRetryInterval key, multiplied by the number of preceding transmissions of this same message.

As an example, consider a Charge Point that has the value "3" for the TransactionMessageAttempts configuration key and the value "60" for the TransactionMessageRetryInterval configuration key. It sends a StopTransaction message and detects a failure to process the message in the Central System. The Charge Point SHALL wait for 60 seconds, and resend the message. In the case when there is a second failure, the Charge Point SHALL wait for 120 seconds, before resending the message. If this final attempt fails, the Charge Point SHOULD discard the message and continue with the next transaction-related message, if there is any.

3.8. Connector numbering

This section is normative.

To enable Central System to be able to address all the connectors of a Charge Point, ConnectorIds MUST always be numbered in the same way.

Connectors numbering (ConnectorIds) MUST be as follows:

- ID of the first connector MUST be 1

- Additional connectors MUST be sequentially numbered (no numbers may be skipped)
- ConnectorIds MUST never be higher than the total number of connectors of a Charge Point
- For operations intiated by the Central System, ConnectorId 0 is reserved for addressing the entire Charge Point.
- For operations initiated by the Charge Point (when reporting), ConnectorId 0 is reserved for the Charge Point main controller.

Example: A Charge Point with 3 connectors: All connectors MUST be numbered with the IDs: 1, 2 and 3. It is advisable to number the connectors of a Charge Point in a logical way: from left to right, top to bottom incrementing.

3.9. ID Tokens

This section is normative.

In most cases, IdToken data acquired via local token reader hardware is usually a (4 or 7 byte) UID value of a physical RFID card, typically represented as 8/14 hexadecimal digit characters.

However, IdTokens sent to Charge Points by Central Systems for remotely initiated charging sessions may commonly be (single use) virtual transaction authorization codes, or virtual RFID tokens that deliberately use a non-standard UID format to avoid possible conflict with real UID values.

Also, IdToken data used as ParentIds may often use a shared central account identifier for the ParentId, instead of a UID of the first/master RFID card of an account.

Therefore, message data elements of the IdToken class (including ParentId) MAY contain any data, subject to the constraints of the data-type (CiString20Type), that is meaningful to a Central System (e.g. for the purpose of identifying the initiator of charging activity), and Charge Points MUST NOT make any presumptions as to the format or content of such data (e.g. by assuming that it is a UID-like value that must be hex characters only and/or an even number of digits).

To promote interoperability, based on common practice to date in the case of IdToken data representing physical ISO 14443 compatible RFID card UIDs, it is RECOMMENDED that such UIDs be represented as hex representations of the UID bytes. According to ISO14443-3, byte 0 should come first in the hex string.

3.10. Parent idTag

This section is normative.

A Central System has the ability to treat a set of identity tokens as a "group", thereby allowing any one token in the group to start a transaction and for the same token, or another token in the same group, to stop the transaction. This supports the common use-cases of families or businesses with multiple drivers using one or more shared electric vehicles on a single recharging contract account.

Tokens (idTags) are grouped for authorization purposes by specifying a common group identifier in the optional ParentId element in IdTagInfo: two idTags are considered to be in the same group if their ParentId Tags match.

Even though the ParentId has the same nominal data type (IdToken) as an idTag, the value of this element may not be in the common format of IdTokens and/or may not represent an actual valid IdToken (e.g. it may be a common shared "account number"): therefore, the ParentId value SHOULD NOT be used for comparison against a presented Token value (unless it also occurs as an idTag value).

3.11. Reservations

This section is informative.

Reservation of a Charge Point is possible using the Reserve Now operation. This operation reserves the Charge Point until a certain expiry time for a specific idTag. A parent idTag may be included in the reservation to support 'group' reservations. It is possible to reserve a specific connector on a Charge Point or to reserve any connector on a Charge Point. A reservation is released when the reserved idTag is used on the reserved connector (when specified) or on any connector (when unspecified) or when the expiry time is

reached or when the reservation is explicitly canceled.

3.12. Vendor-specific data transfer

This section is informative.

The mechanism of vendor-specific data transfer allows for the exchange of data or messages not standardized in OCPP. As such, it offers a framework within OCPP for experimental functionality that may find its way into future OCPP versions. Experimenting can be done without creating new (possibly incompatible) OCPP dialects. Secondly, it offers a possibility to implement additional functionality agreed upon between specific Central System and Charge Point vendors.

The operation Vendor Specific Data MAY be initiated either by the Central System or by the Charge Point.

Please use with extreme caution and only for optional functionality, since it will impact your compatibility with other systems that do not make use of this option. We recommend mentioning the usage explicitly in your documentation and/or communication. Please consider consulting the Open Charge Alliance before turning to this option to add functionality.

3.13. Smart Charging

This section is normative.

With Smart Charging a Central System gains the ability to influence the charging power or current of a specific EV, or the total allowed energy consumption on an entire Charge Point / a group of Charge Points, for instance based on a grid connection, energy availability on the gird or the wiring of a building. Influencing the charge power or current is based on energy transfer limits at specific points in time. Those limits are combined in a Charging Profile.

3.13.1. Charging profile purposes

A charging profile consists of a charging schedule, which is basically a list of time intervals with their maximum charge power or current, and some values to specify the time period and recurrence of the schedule.

There are three different types of charging profiles, depending on their purpose:

- *ChargePointMaxProfile*

In load balancing scenarios, the Charge Point has one or more local charging profiles that limit the power or current to be shared by all connectors of the Charge Point. The Central System SHALL configure such a profile with ChargingProfilePurpose set to "*ChargePointMaxProfile*". *ChargePointMaxProfile* can only be set at Charge Point ConnectorId 0.

- *TxDefaultProfile*

Default schedules for new transactions MAY be used to impose charging policies. An example could be a policy that prevents charging during the day. For schedules of this purpose, ChargingProfilePurpose SHALL be set to *TxDefaultProfile*.

If TxDefaultProfile is set to ConnectorId 0, the TxDefaultProfile is applicable to all Connectors.

If ConnectorId is set >0, it only applies to that specific connector.

In the event a TxDefaultProfile for connector 0 is installed, and the Central System sends a new profile with ConnectorId >0, the TxDefaultProfile SHALL be replaced only for that specific connector.

- *TxProfile*

If a transaction-specific profile with purpose *TxProfile* is present, it SHALL overrule the default charging profile with purpose TxDefaultProfile for the duration of the current transaction only. After the transaction is stopped the profile SHOULD be deleted. If there is no transaction active on the connector specified in a charging profile of type *TxProfile*, then the Charge Point SHALL discard it and return an error status in SetChargingProfile.conf.

The final schedule constraints that apply to a transaction are determined by merging the profiles with purposes *ChargePointMaxProfile* with the profile *TxProfile* or the TxDefaultProfile in case no profile of purpose *TxProfile* is provided. *TxProfile SHALL only be set at Charge Point ConnectorId >0*.

3.13.2. Stacking charging profiles

It is allowed to stack charging profiles of the same charging profile purpose in order to describe complex calendars. For example, one can define a charging profile of purpose TxDefaultProfile with a duration and recurrence of one week that allows full power or current charging on weekdays from 23:00h to 06:00h and from 00:00h to 24:00h in weekends and reduced power or current charging at other times. On top of that, one can define other TxDefaultProfiles that define exception to this rule, for example for holidays.

Precedence of charging profiles is determined by the value of their StackLevel parameter. At any point in time the prevailing charging profile SHALL be the charging profile with the highest stackLevel among the profiles that are valid at that point in time, as determined by their validFrom and validTo parameters.

To avoid conflicts, the existence of multiple Charging Profiles with the same stackLevel and Purposes in a Charge Point is not allowed. Whenever a Charge Point receives a Charging Profile with a stackLevel and Purpose that already exists in the Charge Point, the Charge Point SHALL replace the existing profile.

In the case an updated charging profile (with the same stackLevel and purpose) is sent with a validFrom dateTime in the future, the Charge Point SHALL replace the installed profile and SHALL revert to default behavior until validFrom is reached. It is RECOMMENDED to provide a start time in the past to prevent gaps.

If you use Stacking without a duration, on the highest stack level, the Charge Point will never fall back to a lower stack level profile.

3.13.3. Combining charging profile purposes

The Composite Schedule that will guide the charging level is a combination of the prevailing Charging Profiles of the different chargingProfilePurposes.

This Composite Schedule is calculated by taking the minimum value for each time interval. Note that time intervals do not have to be of fixed length, nor do they have to be the same for every charging profile purpose.
This means that a resulting Composite Schedule MAY contain intervals of different lengths.

At any point in time, the available power or current in the Composite Schedule, which is the result of merging the schedules of charging profiles ChargePointMaxProfile and TxDefaultProfile (or TxProfile), SHALL be less than or equal to lowest value of available power or current in any of the merged schedules.

In the case the Charge Point is equipped with more than one Connector, the limit value of ChargePointMaxProfile is the limit for all connectors combined. The combined energy flow of all connectors SHALL NOT be greater then the limit set by ChargePointMaxProfile.

3.13.4. Smart Charging Use Cases

This section is informative.

There may be many different uses for smart charging. The following three typical kinds of smart charging will be used to illustrate the possible behavior of smart charging:

- Load balancing
- Central smart charging
- Local smart charging

There are more complex use cases possible in which two or more of the above use cases are combined into one more complex system.

Load Balancing

This section is informative.

The Load Balancing use case is about internal load balancing within the Charge Point, the Charge Point controls the charging schedule per connector. The Charge Point is configured with a fixed limit, for example the maximum current of the connection to the grid.

The optional charging schedule field minChargingRate may be used by the Charge Point to optimize the power distribution between the connectors. The parameter informs the Charge Point that charging below minChargingRate is inefficient, giving the possibility to select another balancing strategy.

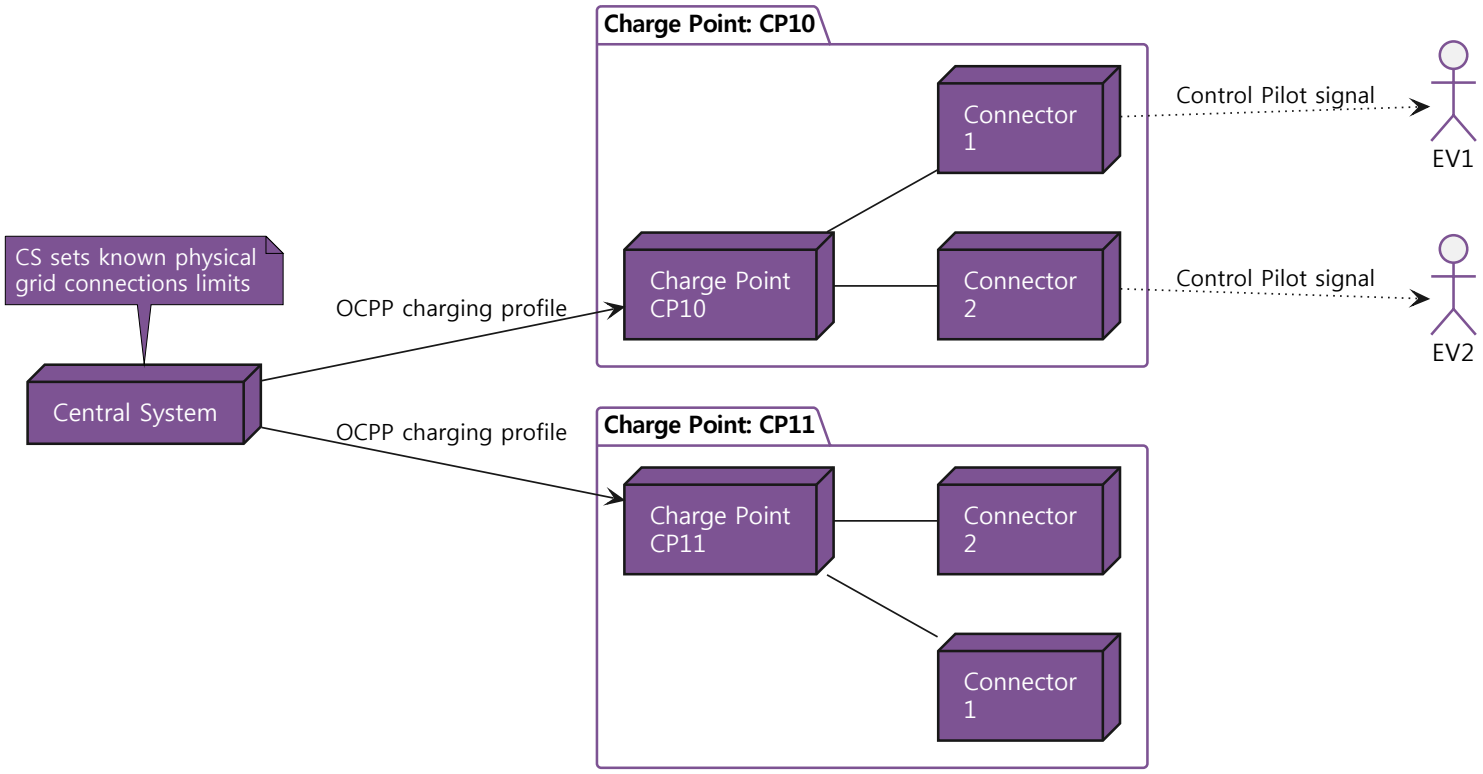

Figure 6. Load balancing Smart Charging topology

Central Smart Charging

This section is informative.

With Central smart charging the constraints on the charging schedule, per transaction, are determined by the Central System. The Central System uses these schedules to stay within limits imposed by any external system.

The Central System directly controls the limits on the connectors of the Charge Points.

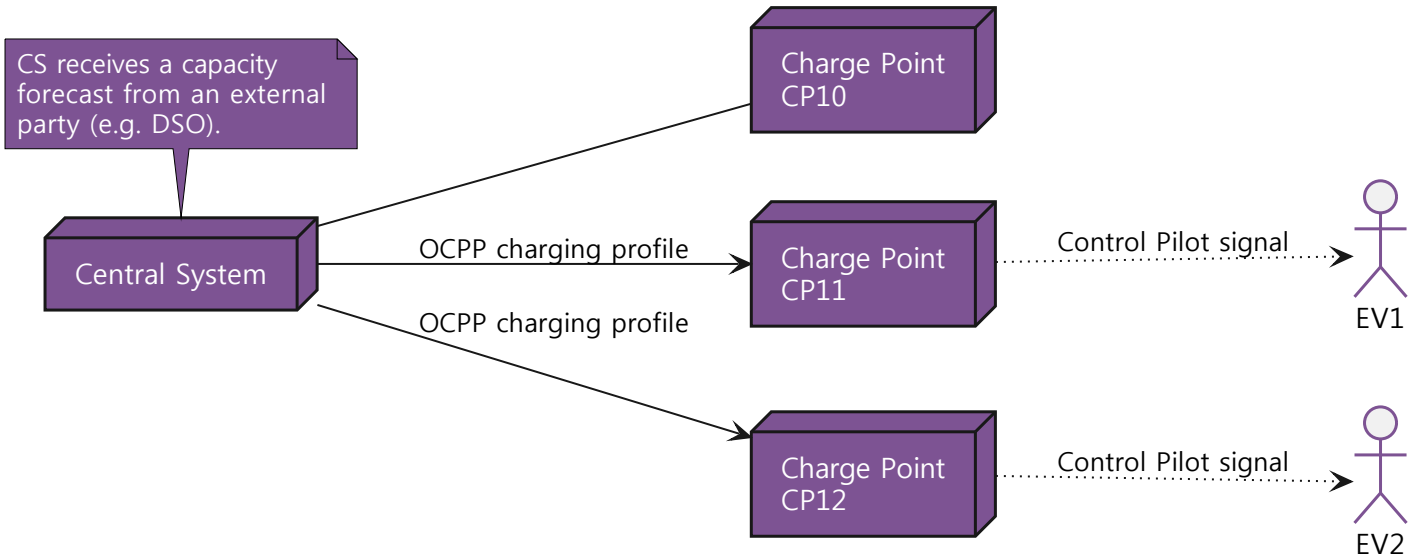

Figure 7. Central Smart Charging topology

Central smart charging assumes that charge limits are controlled by the Central System. The Central System receives a capacity forecast from the grid operator (DSO) or another source in one form or another and calculates charging schedules for some or all charging transactions, details of which are out of scope of this specification.

The Central System imposes charging limits on connectors. In response to a StartTransaction.req PDU The Central System may choose to set charging limits to the transaction using the TxProfile.

Central Smart Charging can be done with a Control Pilot signal, albeit with some limitations, because an EV cannot communicate its charging via the Control Pilot signal. In analogy to the Local Smart Charging use case, a connector can execute a charging schedule by the Control Pilot signal. This is illustrated in the Figure below:

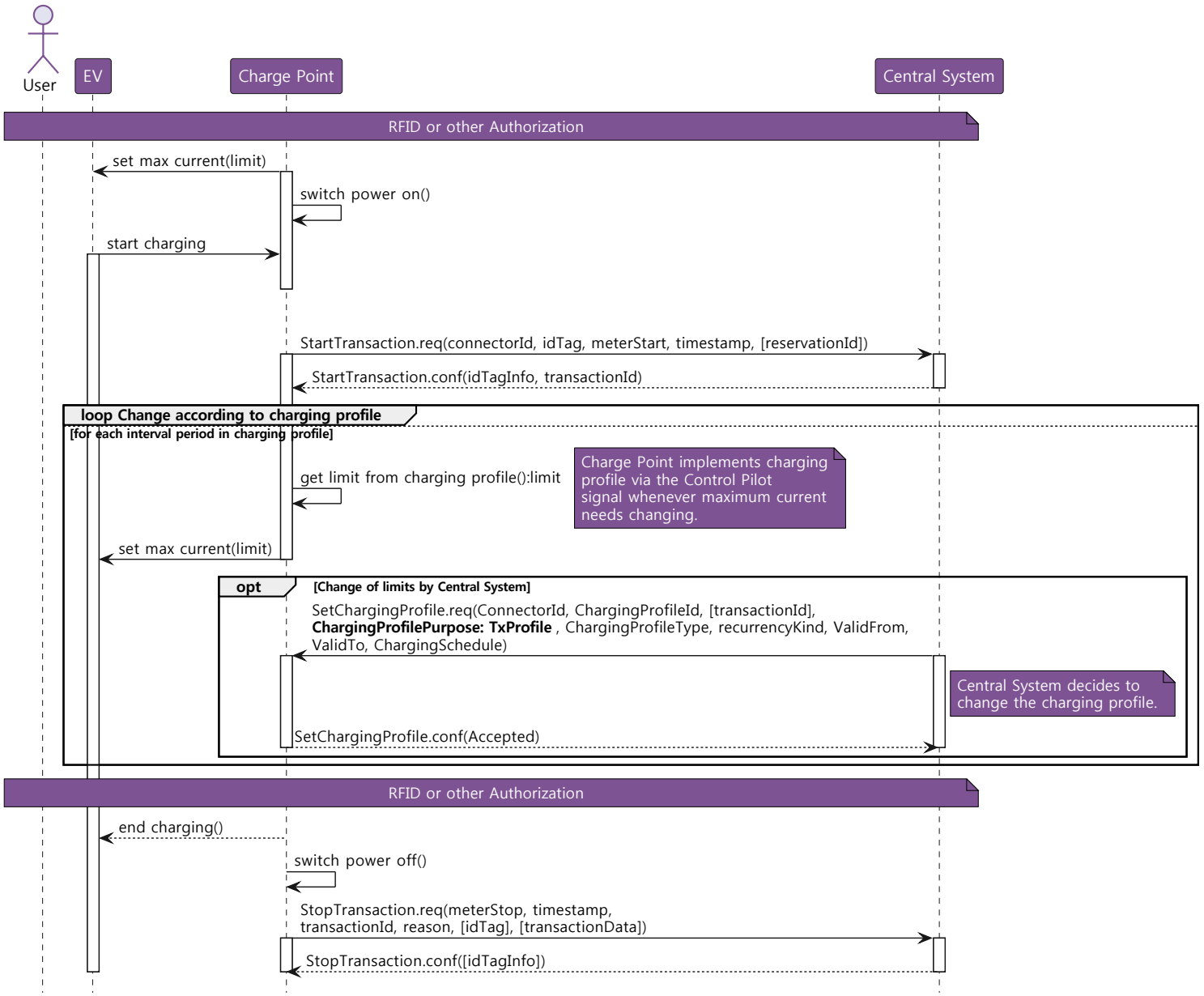

Figure 8. Sequence Diagram: Central Smart Charging

Explanation for the above figure:

- After authorization the connector will set a maximum current to use via the Control Pilot signal. This limit is based on a (default) charging profile that the connector had previously received from the Central System. The EV starts charging and a StartTransaction.req is sent to the Central System.
- While charging is in progress the connector will continuously adapt the maximum current or power according to the charging profile. Optionally, at any point in time the Central System may send a new charging profile for the connector that shall be used as a limit schedule for the EV.

Local Smart Charging

The Local Smart Charging use case describes a use case in which smart charging enabled Charge Points have charging limits controlled locally by a Local Controller, not the Central System. The use case for local smart charging is about limiting the amount of power that can be used by a group of Charge Points, to a certain maximum. A typical use would be a number of Charge Points in a parking garage where the rating of the connection to the grid is less than the sum the ratings of the Charge Points. Another application might be that the Local Controller receives information about the availability of power from a DSO or a local smart grid node.

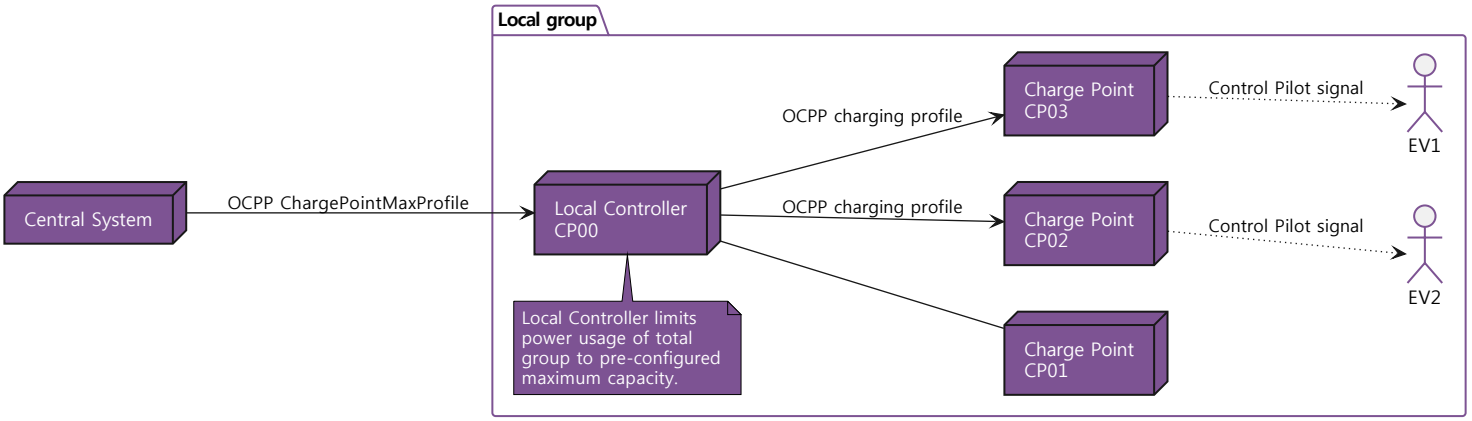

Figure 9. Local Smart Charging topology

Local smart charging assumes the existence of a Local Controller to control a group of Charge Points. The Local Controller is a logical component. It may be implemented either as a separate physical component or as part of a 'master' Charge Point controlling a number of other Charge Points. The Local Control implements the OCPP protocol and is a proxy for the group members' OCPP messages, and may or may not have any connectors of its own.

In the case of local smart charging the Local Controller imposes charging limits on a Charge Point. These limits may be changed dynamically during the charging process in order to keep the power consumption of the group of Charge Points within the group limits. The group limits may be pre-configured in the Local Controller or may have been configured by the Central System.

The optional charging schedule field minChargingRate may be used by the Local Controller to optimize the power distribution between the connectors. The parameter informs the Local Controller that charging below minChargingRate is inefficient, giving the possibility to select another balancing strategy.

The following diagram illustrates the sequence of messages to set charging limits on Charge Points in a Local Smart Charging group. These limits can either be pre-configured in the Local Controller in one way or another, or they can be set by the Central System. The Local Controller contains the logic to distribute this capacity among the connected connectors by adjusting their limits as needed.

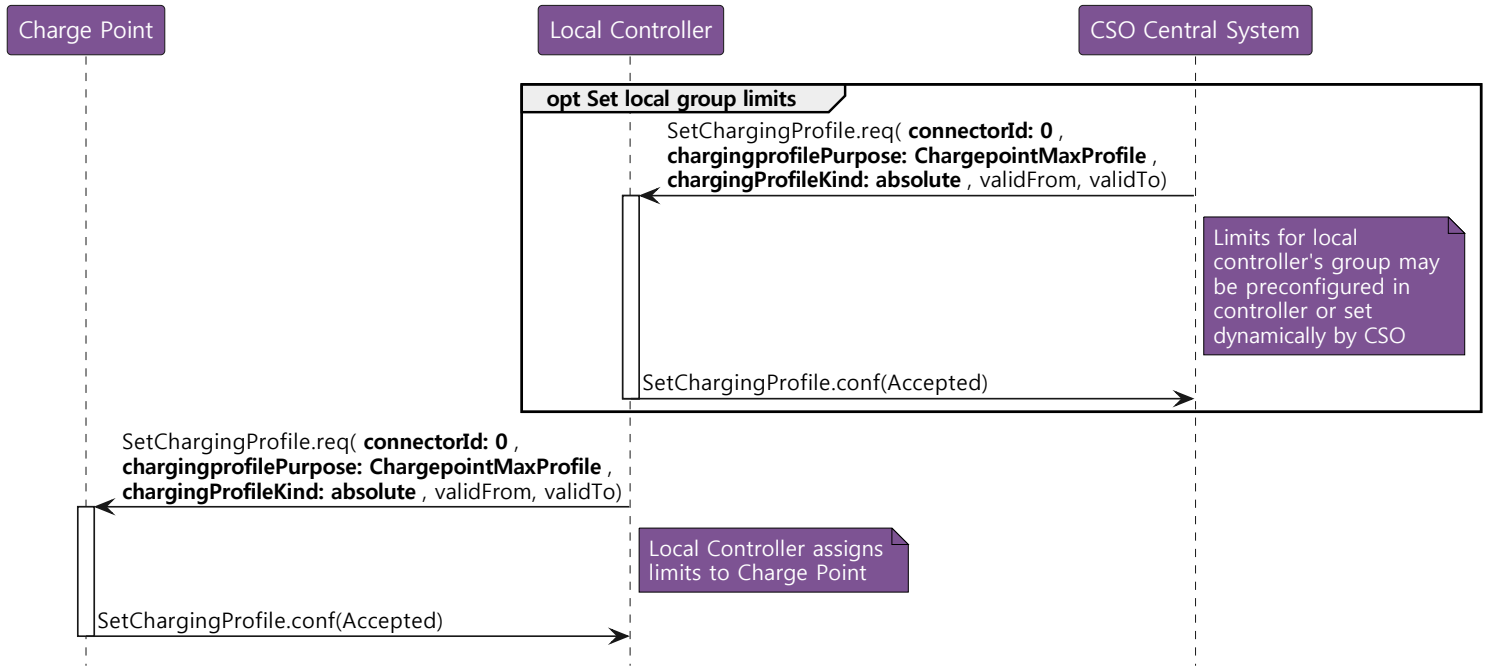

Figure 10. Presetting Local Group Limits

The next diagram describe the sequence of messages for a typical case of Local Smart Charging. For simplicity's sake, this case only involves one connector.

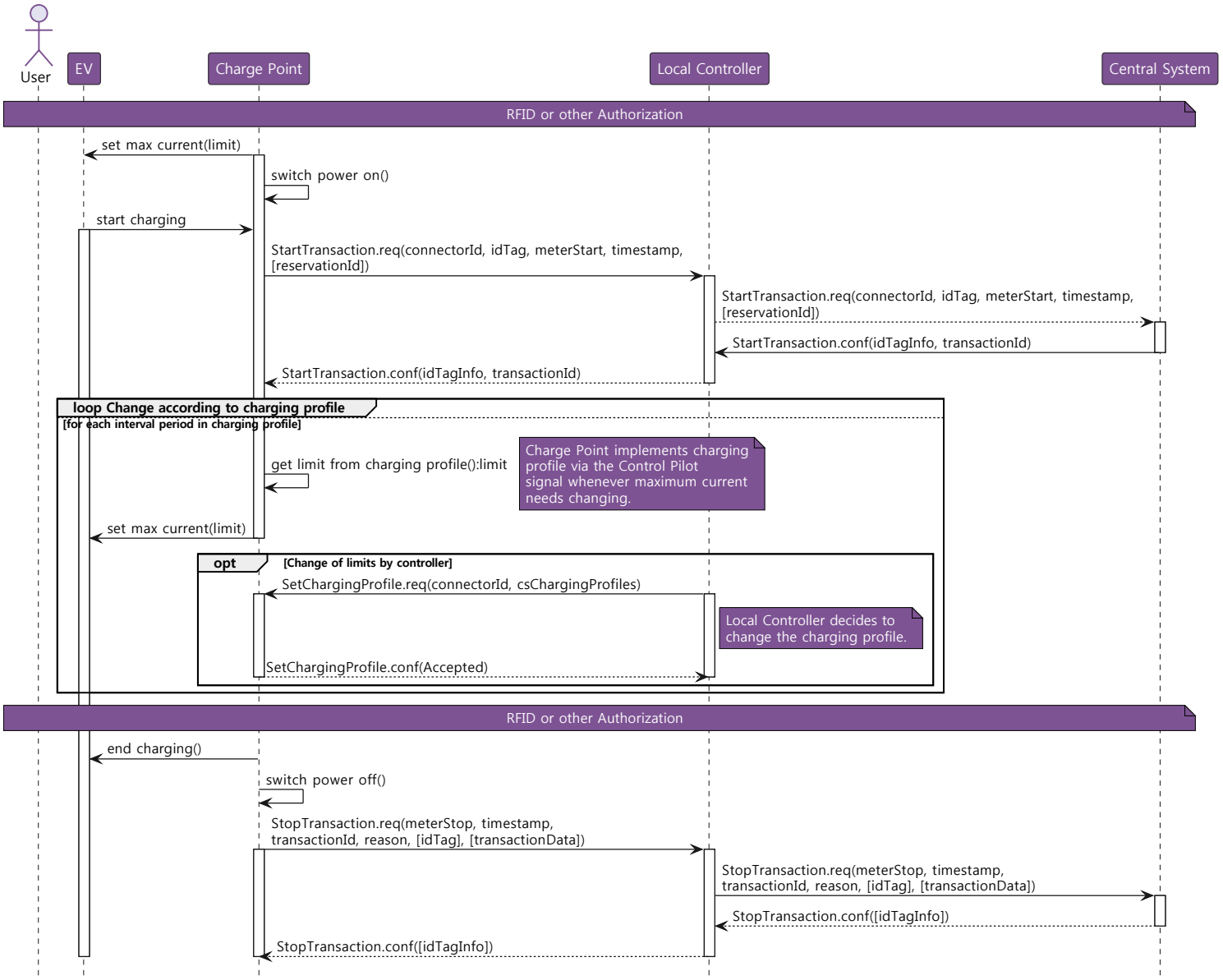

Figure 11. Sequence Diagram: Local Smart Charging

Explanation for the above figure:

- After authorization the connector will set a maximum current to use, via the Control Pilot signal. This limit is based on a (default) charging profile that the connector had previously received from the Local Controller. The EV starts charging and sends a StartTransaction.req.
- The StartTransaction.req is sent to the Central System via the Local Controller, so that also the Local Controller knows a transaction has started. The Local Controller just passes on the messages between Charge Point and Central System, so that the Central System can address all the Local Smart Charging group members individually.
- While charging is in progress the connector will continuously adapt the maximum current according to the charging profile. Optionally, at any point in time the Local Controller may send a new charging profile to the connector that shall be used as a limit schedule for the EV.

3.13.5. Discovery of Charge Point Capabilities

This section is normative.

The smart charging options defined can be used in extensive ways. Because of the possible limitations and differences in capabilities between Charge Points, the Central System needs to be able to discover the Charge Point specific capabilities. This is ensured by the standardized configuration keys as defined in this chapter. A Smart Charging enabled Charge Point SHALL implement, and support reporting of, the following configuration keys through the GetConfiguration.req PDU

SMART CHARGING CONFIGURATION KEYS
ChargeProfileMaxStackLevel
ChargingScheduleAllowedChargingRateUnit
ChargingScheduleMaxPeriods
MaxChargingProfilesInstalled

A full list of all standardized configuration keys can be found in chapter Standard Configuration Key Names & Values.

3.13.6. Offline behavior of smart charging

This section is normative.

If a Charge Point goes *offline* after having received a transaction-specific charging profile with purpose TxProfile, then it SHALL continue to use this profile for the duration of the transaction.

If a Charge Point goes *offline* before a transaction is started or before a transaction-specific charging profile with purpose TxProfile was received, then it SHALL use the charging profiles that are available. Zero or more of the following charging profile purposes MAY have been previously received from the Central System:

- *ChargePointMaxProfile*
- *TxDefaultProfile*

See section Combining Charging Profile Purposes for a description on how to combine charging profiles with different purposes.

If a Charge Point goes *offline* , without having any charging profiles, then it SHALL execute a transaction as if no constraints apply.

3.13.7. Example data structure for smart charging

This section is informative.

The following data structure describes a daily default profile that limits the power to 6 kW between 08:00h and 20:00h.

CHARGINGPROFILE		
chargingProfileId	**100**	
stackLevel	**0**	
chargingProfilePurpose	**TxDefaultProfile**	
chargingProfileKind	**Recurring**	
recurrencyKind	**Daily**	
chargingSchedule	(List of 1 ChargingSchedule elements)	
	ChargingSchedule	
	duration	**86400 (= 24 hours)**

CHARGINGPROFILE			
	startSchedule	**2013-01-01T00:00Z**	
	chargingRateUnit	**W**	
	chargingSchedulePeriod	(List of 3 ChargingSchedulePeriod elements)	
		ChargingSchedulePeriod	
		startPeriod	**0 (=00:00)**
		limit	**11000**
		numberPhases	3
		startPeriod	**28800 (=08:00)**
		limit	**6000**
		numberPhases	3
		startPeriod	**72000 (=20:00)**
		limit	**11000**
		numberPhases	3

The amount of phases used during charging is limited by the capabilities of: The Charge Point, EV and Cable between CP and EV. If any of these 3 is not capable of 3 phase charging, the EV will be charged using 1 phase only.

Switching the number of used phases during a schedule or charging session should be done with care. Some EVs may not support this and changing the amount of phases may result in physical damage. With the configuration key: ConnectorSwitch3to1PhaseSupported The Charge Point can tell if it supports switching the amount of phases during a transaction.

On days on which DST goes into or out of effect, a special profile might be needed (e.g. for relative profiles)

3.14. Time zones

This section is informative.

OCPP does not prescribe the use of a specific time zone for time values. However, it is strongly recommended to use UTC for all time values to improve interoperability between Central Systems and Charge Points.

3.15. Time notations

This section is normative.

Implementations MUST use ISO 8601 date time notation. Message receivers must be able to handle fractional seconds and time zone offsets (another implementation might use them). Message senders MAY save data usage by omitting insignificant fractions of seconds.

3.16. Metering Data

This section is normative.

Extensive metering data relating to charging sessions can be recorded and transmitted in different ways depending on its intended purpose. There are two obvious use cases (but the use of meter values is not limited to these two):

- Charging Session Meter Values
- Clock-Aligned Meter Values

Both types of meter readings MAY be reported in standalone MeterValues.req messages (during a transaction) and/or as part of the transactionData element of the StopTransaction.req PDU.

3.16.1. Charging Session Meter Values

Frequent (e.g. 1-5 minute interval) meter readings taken and transmitted (usually in "real time") to the Central System, to allow it to provide information updates to the EV user (who is usually not at the charge point), via web, app, SMS, etc., as to the progress of the charging session. In OCPP, this is called "sampled meter data", as the exact frequency and time of readings is not very significant, as long as it is "frequent enough". "Sampled meter data" can be configured with the following configuration keys:

- `MeterValuesSampledData`
- `MeterValuesSampledDataMaxLength`
- `MeterValueSampleInterval`
- `StopTxnSampledData`
- `StopTxnSampledDataMaxLength`

`MeterValueSampleInterval` is the time (in seconds) between sampling of metering (or other) data, intended to be transmitted by "MeterValues" PDUs. Samples are acquired and transmitted periodically at this interval from the start of the charging transaction.

A value of "0" (numeric zero), by convention, is to be interpreted to mean that no sampled data should be transmitted.

`MeterValuesSampledData` is a comma separated list that prescribes the set of measurands to be included in a MeterValues.req PDU, every `MeterValueSampleInterval` seconds. The maximum amount of elements in the `MeterValuesSampledData` list can be reported by the Charge Point via: `MeterValuesSampledDataMaxLength`

`StopTxnSampledData` is a comma separated list that prescribes the sampled measurands to be included in the TransactionData element of StopTransaction.req PDU, every `MeterValueSampleInterval` seconds from the start of the Transaction. The maximum amount of elements in the `StopTxnSampledData` list can be reported by the Charge Point via: `StopTxnSampledDataMaxLength`.

3.16.2. Clock-Aligned Meter Values

Grid Operator might require meter readings to be taken from fiscally certified energy meters, at specific Clock aligned times (usually every quarter hour, or half hour).

"Clock-Aligned Billing Data" can be configured with the following configuration keys:

- ClockAlignedDataInterval
- MeterValuesAlignedData
- MeterValuesAlignedDataMaxLength
- StopTxnAlignedData
- StopTxnAlignedDataMaxLength

ClockAlignedDataInterval is the size of the clock-aligned data interval (in seconds). This defines the set of evenly spaced meter data aggregation intervals per day, starting at 00:00:00 (midnight).

For example, a value of 900 (15 minutes) indicates that every day should be broken into 96 15-minute intervals.

A value of "0" (numeric zero), by convention, is to be interpreted to mean that no clock-aligned data should be transmitted.

MeterValuesAlignedData is a comma separated list that prescribes the set of measurands to be included in a MeterValues.req PDU, every ClockAlignedDataInterval seconds. The maximum amount of elements in the MeterValuesAlignedData list can be reported by the Charge Point via:

MeterValuesAlignedDataMaxLength

StopTxnAlignedData is a comma separated list that prescribes the set of clock-aligned periodic measurands to be included in the TransactionData element of StopTransaction.req PDU for every ClockAlignedDataInterval of the Transaction. The maximum amount of elements in the StopTxnAlignedData list can be reported by the Charge Point via: StopTxnAlignedDataMaxLength

3.16.3. Multiple Locations/Phases

When a Charge Point can measure the same measurand on multiple locations or phases, all possible locations and/or phases SHALL be reported when configured in one of the relevant configuration keys.

For example: A Charge Point capable of measuring *Current.Import* on *Inlet* (all 3 phases) (grid connection) and *Outlet* (3 phases per connector on both its connectors). *Current.Import* is set in MeterValuesSampledData. MeterValueSampleInterval is set to 300 (seconds). Then the Charge Point should send:

- a MeterValues.req with: *connectorId* = 0; with 3 *SampledValue* elements, one per phase with *location* = *Inlet*.
- a MeterValues.req with: *connectorId* = 1; with 3 *SampledValue* elements, one per phase with *location* = *Outlet*.
- a MeterValues.req with: *connectorId* = 2; with 3 *SampledValue* elements, one per phase with *location* = *Outlet*.

3.16.4. Unsupported measurands

When a Central System sends a ChangeConfiguration.req to a Charge Point with one of the following configuration keys:

- MeterValuesAlignedData
- MeterValuesSampledData
- StopTxnAlignedData
- StopTxnSampledData

If the comma separated list contains one or more measurands that are not supported by this Charge Point, the Charge Point SHALL respond with: ChangeConfiguration.conf with: *status* = *Rejected*. No changes SHALL be made to the currently configuration.

3.16.5. No metering data in a Stop Transaction

When the configuration keys: StopTxnAlignedData and StopTxnSampledData are set to an empty string, the Charge Point SHALL not put meter values in a StopTransaction.req PDU.

Chapter 4. Operations Initiated by Charge Point

4.1. Authorize

Figure 12. Sequence Diagram: Authorize

Before the owner of an electric vehicle can start or stop charging, the Charge Point has to authorize the operation. The Charge Point SHALL only supply energy after authorization. When stopping a Transaction, the Charge Point SHALL only send an Authorize.req when the identifier used for stopping the transaction is different from the identifier that started the transaction.

Authorize.req SHOULD only be used for the authorization of an identifier for charging.

A Charge Point MAY authorize identifier locally without involving the Central System, as described in Local Authorization List. If an idTag presented by the user is not present in the Local Authorization List or Authorization Cache, then the Charge Point SHALL send an Authorize.req PDU to the Central System to request authorization. If the idTag is present in the Local Authorization List or Authorization Cache, then the Charge Point MAY send an Authorize.req PDU to the Central System.

Upon receipt of an Authorize.req PDU, the Central System SHALL respond with an Authorize.conf PDU. This response PDU SHALL indicate whether or not the idTag is accepted by the Central System. If the Central System accepts the idTag then the response PDU MAY include a **parentIdTag** and MUST include an authorization status value indicating acceptance or a reason for rejection.

If Charge Point has implemented an Authorization Cache, then upon receipt of an Authorize.conf PDU the Charge Point SHALL update the cache entry, if the idTag is not in the Local Authorization List, with the IdTagInfo value from the response as described under Authorization Cache.

4.2. Boot Notification

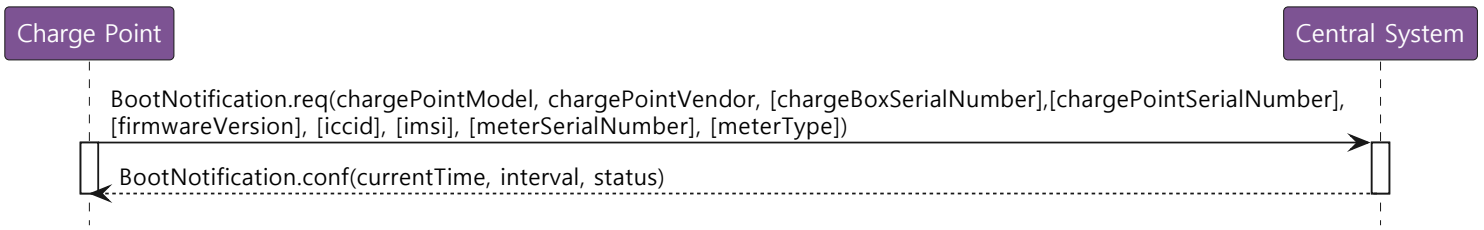

Figure 13. Sequence Diagram: Boot Notification

After start-up, a Charge Point SHALL send a request to the Central System with information about its configuration (e.g. version, vendor, etc.). The Central System SHALL respond to indicate whether it will accept the Charge Point.

The Charge Point SHALL send a BootNotification.req PDU each time it boots or reboots. Between the physical power-on/reboot and the successful completion of a BootNotification, where Central System returns *Accepted* or *Pending* , the Charge Point SHALL NOT send any other request to the Central System. This includes cached messages that are still present in the Charge Point from before.

When the Central System responds with a BootNotification.conf with a status *Accepted* , the Charge Point will adjust the heartbeat interval in accordance with the interval from the response PDU and it is RECOMMENDED to synchronize its internal clock with the supplied Central System's current time. If the Central System returns something other than *Accepted* , the value of the interval field indicates the minimum wait time before sending a next BootNotification request. If that interval value is zero, the Charge Point chooses a waiting interval on its own, in a way that avoids flooding the Central System with requests. A Charge Point SHOULD NOT send a BootNotification.req earlier, unless requested to do so with a TriggerMessage.req.

If the Central System returns the status *Rejected* , the Charge Point SHALL NOT send any OCPP message to the Central System until the aforementioned retry interval has expired. During this interval the Charge Point may no longer be reachable from the Central System. It MAY for instance close its communication channel or shut down its communication hardware. Also the Central System MAY close the communication channel, for instance to free up system resources. While *Rejected* , the Charge Point SHALL NOT respond to any Central System initiated message. the Central System SHOULD NOT initiate any.

The Central System MAY also return a *Pending* registration status to indicate that it wants to retrieve or set certain information on the Charge Point before the Central System will accept the Charge Point. If the Central System returns the *Pending* status, the communication channel SHOULD NOT be closed by either the Charge Point or the Central System. The Central System MAY send request messages to retrieve information from the Charge Point or change its configuration. The Charge Point SHOULD respond to these messages. The Charge Point SHALL NOT send request messages to the Central System unless it has been instructed by the Central System to do so with a TriggerMessage.req request.

While in *pending* state, the following Central System initiated messages are not allowed: RemoteStartTransaction.req and RemoteStopTransaction.req

4.2.1. Transactions before being accepted by a Central System

A Charge Point Operator MAY choose to configure a Charge Point to accept transactions before the Charge Point is accepted by a Central System. Parties who want to implement this such behavior should realize that it is uncertain if those transactions can ever be delivered to the Central System.

After a restart (for instance due to a remote reset command, power outage, firmware update, software error etc.) the Charge Point MUST again contact the Central System and SHALL send a BootNotification request. If the Charge Point fails to receive a BootNotification.conf from the Central System, and has no in-built non-volatile real-time clock hardware that has been correctly preset, the Charge Point may not have a valid date / time setting, making it impossible to later determine the date / time of transactions.

It might also be the case (e.g. due to configuration error) that the Central System indicates a status other than Accepted for an extended period of time, or indefinitely.

It is usually advisable to deny all charging services at a Charge Point if the Charge Point has never before been Accepted by the Central System (using the current connection settings, URL, etc.) since users cannot be authenticated and running transactions could conflict with provisioning processes.

4.3. Data Transfer

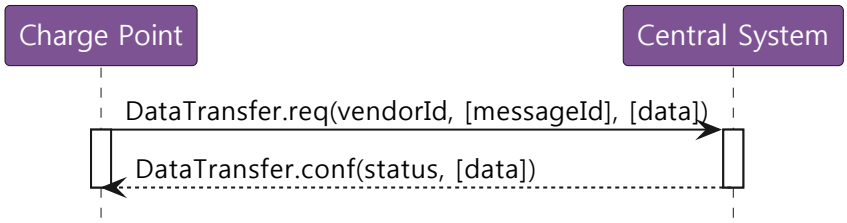

Figure 14. Sequence Diagram: Data Transfer

If a Charge Point needs to send information to the Central System for a function not supported by OCPP, it SHALL use the DataTransfer.req PDU.

The vendorId in the request SHOULD be known to the Central System and uniquely identify the vendor-specific implementation. 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 organisation.

Optionally, the messageId in the request PDU MAY be used to indicate a specific message or implementation.

The length of data in both the request and response PDU is undefined and should be agreed upon by all parties involved.

If the recipient of the request has no implementation for the specific vendorId it SHALL return a status 'UnknownVendor' and the data element SHALL not be present. In case of a messageId mismatch (if used) the recipient SHALL return status 'UnknownMessageId'. In all

other cases the usage of status 'Accepted' or 'Rejected' and the data element is part of the vendor-specific agreement between the parties involved.

4.4. Diagnostics Status Notification

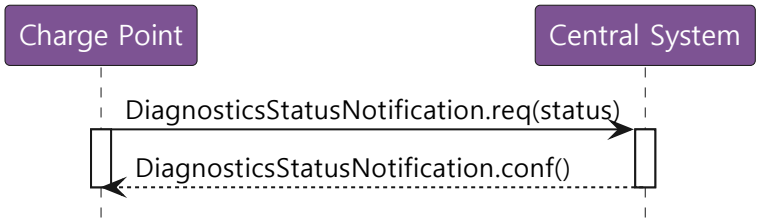

Figure 15. Sequence Diagram: Diagnostics Status Notification

Charge Point sends a notification to inform the Central System about the status of a diagnostics upload. The Charge Point SHALL send a DiagnosticsStatusNotification.req PDU to inform the Central System that the upload of diagnostics is busy or has finished successfully or failed. The Charge Point SHALL only send the status Idle after receipt of a TriggerMessage for a Diagnostics Status Notification, when it is not busy uploading diagnostics.

Upon receipt of a DiagnosticsStatusNotification.req PDU, the Central System SHALL respond with a DiagnosticsStatusNotification.conf.

4.5. Firmware Status Notification

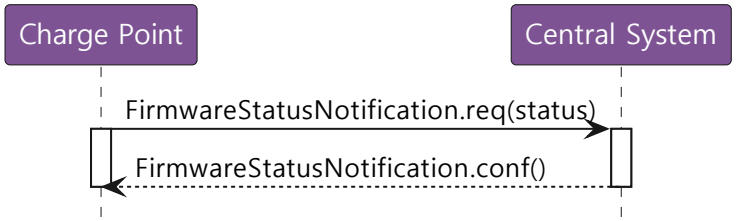

Figure 16. Sequence Diagram: Firmware Status Notification

A Charge Point sends notifications to inform the Central System about the progress of the firmware update. The Charge Point SHALL send a FirmwareStatusNotification.req PDU for informing the Central System about the progress of the downloading and installation of a firmware update. The Charge Point SHALL only send the status Idle after receipt of a TriggerMessage for a Firmware Status Notification, when it is not busy downloading/installing firmware.

Upon receipt of a FirmwareStatusNotification.req PDU, the Central System SHALL respond with a FirmwareStatusNotification.conf.

The FirmwareStatusNotification.req PDUs SHALL be sent to keep the Central System updated with the status of the update process, started by the Central System with a FirmwareUpdate.req PDU.

4.6. Heartbeat

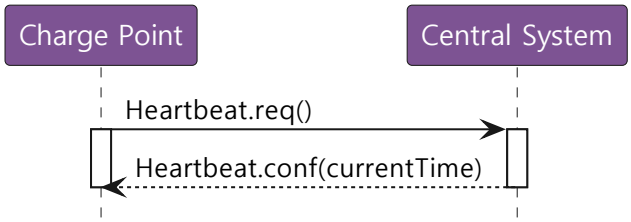

Figure 17. Sequence Diagram: Heartbeat

To let the Central System know that a Charge Point is still connected, a Charge Point sends a heartbeat after a configurable time interval.

The Charge Point SHALL send a Heartbeat.req PDU for ensuring that the Central System knows that a Charge Point is still alive.

Upon receipt of a Heartbeat.req PDU, the Central System SHALL respond with a Heartbeat.conf. The response PDU SHALL contain the current time of the Central System, which is RECOMMENDED to be used by the Charge Point to synchronize its internal clock.

The Charge Point MAY skip sending a Heartbeat.req PDU when another PDU has been sent to the Central System within the configured heartbeat interval. This implies that a Central System SHOULD assume availability of a Charge Point whenever a PDU has been received, the same way as it would have, when it received a Heartbeat.req PDU.

With JSON over WebSocket, sending heartbeats is not mandatory. However, for time synchronization it is advised to at least send one heartbeat per 24 hour.

4.7. Meter Values

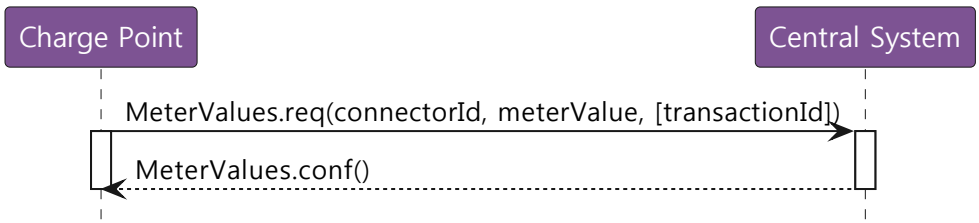

Figure 18. Sequence Diagram: Meter Values

A Charge Point MAY sample the electrical meter or other sensor/transducer hardware to provide extra information about its meter values. It is up to the Charge Point to decide when it will send meter values. This can be configured using the ChangeConfiguration.req message to data acquisition intervals and specify data to be acquired & reported.

The Charge Point SHALL send a MeterValues.req PDU for offloading meter values. The request PDU SHALL contain for each sample:

1. The id of the Connector from which samples were taken. If the connectorId is 0, it is associated with the entire Charge Point. If the connectorId is 0 and the Measurand is energy related, the sample SHOULD be taken from the main energy meter.
2. The transactionId of the transaction to which these values are related, if applicable. If there is no transaction in progress or if the values are taken from the main meter, then transaction id may be omitted.
3. One or more **meterValue** elements, of type MeterValue, each representing a set of one or more data values taken at a particular point in time.

Each MeterValue element contains a timestamp and a set of one or more individual sampledvalue elements, all captured at the same point in time. Each sampledValue element contains a single value datum. The nature of each sampledValue is determined by the optional measurand, context, location, unit, phase, and format fields.

The optional measurand field specifies the type of value being measured/reported.

The optional context field specifies the reason/event triggering the reading.

The optional location field specifies where the measurement is taken (e.g. Inlet, Outlet).

The optional phase field specifies to which phase or phases of the electric installation the value applies. The Charging Point SHALL report all phase number dependent values from the electrical meter (or grid connection when absent) point of view.

The phase field is not applicable to all Measurands.

Two measurands (*Current.Offered* and *Power.Offered*) are available that are strictly speaking no measured values. They indicate the maximum amount of current/power that is being offered to the EV and are intended for use in smart charging applications.

For individual connector phase rotation information, the Central System MAY query the ConnectorPhaseRotation configuration key on the Charging Point via GetConfiguration. The Charge Point SHALL report the phase rotation in respect to the grid connection. Possible values per connector are NotApplicable, Unknown, RST, RTS, SRT, STR, TRS and TSR. see section Standard Configuration Key Names & Values for more information.

The **EXPERIMENTAL** optional format field specifies whether the data is represented in the normal (default) form as a simple numeric value ("**Raw**"), or as "**SignedData**", an opaque digitally signed binary data block, represented as hex data. This experimental field may be deprecated and subsequently removed in later versions, when a more mature solution alternative is provided.

To retain backward compatibility, the default values of all of the optional fields on a sampledValue element are such that a **value** without any additional fields will be interpreted, as a register reading of active import energy in Wh (Watt-hour) units.

Upon receipt of a MeterValues.req PDU, the Central System SHALL respond with a MeterValues.conf.

It is likely that The Central System applies sanity checks to the data contained in a MeterValues.req it received. The outcome of such sanity checks SHOULD NOT ever cause the Central System to not respond with a MeterValues.conf. Failing to respond with a MeterValues.conf will only cause the Charge Point to try the same message again as specified in Error responses to transaction-related messages.

4.8. Start Transaction

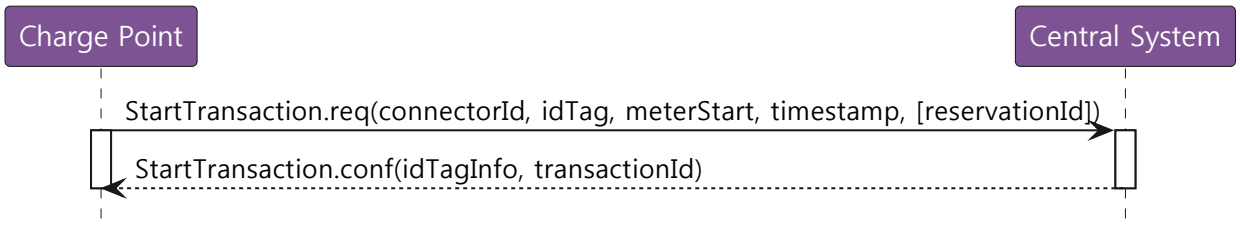

Figure 19. Sequence Diagram: Start Transaction

The Charge Point SHALL send a StartTransaction.req PDU to the Central System to inform about a transaction that has been started. If this transaction ends a reservation (see Reserve Now operation), then the StartTransaction.req MUST contain the reservationId.

Upon receipt of a StartTransaction.req PDU, the Central System SHOULD respond with a StartTransaction.conf PDU. This response PDU MUST include a transaction id and an authorization status value.

The Central System MUST verify validity of the identifier in the StartTransaction.req PDU, because the identifier might have been authorized locally by the Charge Point using outdated information. The identifier, for instance, may have been blocked since it was added to the Charge Point's Authorization Cache.

If Charge Point has implemented an Authorization Cache, then upon receipt of a StartTransaction.conf PDU the Charge Point SHALL update the cache entry, if the idTag is not in the Local Authorization List, with the IdTagInfo value from the response as described under Authorization Cache.

It is likely that The Central System applies sanity checks to the data contained in a StartTransaction.req it received. The outcome of such sanity checks SHOULD NOT ever cause the Central System to not respond with a StartTransaction.conf. Failing to respond with a StartTransaction.conf will only cause the Charge Point to try the same message again as specified in Error responses to transaction-related messages.

4.9. Status Notification

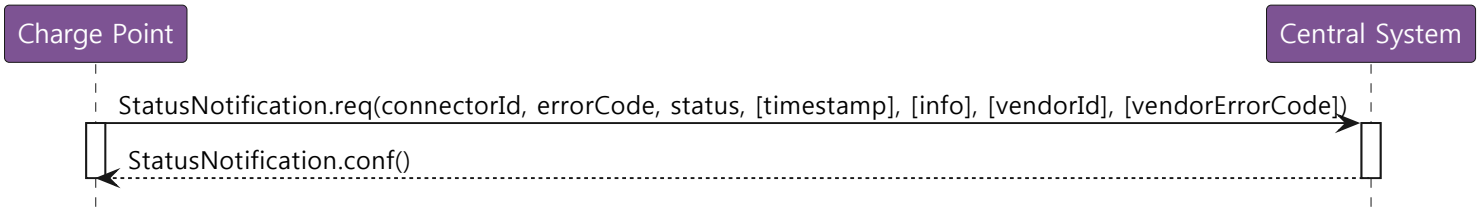

Figure 20. Sequence Diagram: Status Notification

A Charge Point sends a notification to the Central System to inform the Central System about a status change or an error within the Charge Point. The following table depicts changes from a previous status (left column) to a new status (upper row) upon which a Charge Point MAY send a StatusNotification.req PDU to the Central System.

The Occupied state as defined in previous OCPP versions is no longer relevant. The Occupied state is split into five new statuses: *Preparing*, *Charging*, *SuspendedEV*, *SuspendedEVSE* and *Finishing*.

EVSE is used in Status Notification instead of Socket or Charge Point for future compatibility.

The following table describes which status transitions are possible:

	State From / To:	**1** Available	**2** Preparing	**3** Charging	**4** SuspendedEV	**5** SuspendedEVSE	**6** Finishing	**7** Reserved	**8** Unavailable	**9** Faulted
A	Available		A2	A3	A4	A5		A7	A8	A9
B	Preparing	B1		B3	B4	B5	B6			B9
C	Charging	C1			C4	C5	C6		C8	C9
D	SuspendedEV	D1		D3		D5	D6		D8	D9
E	SuspendedEVSE	E1		E3	E4		E6		E8	E9
F	Finishing	F1	F2						F8	F9
G	Reserved	G1	G2						G8	G9
H	Unavailable	H1	H2	H3	H4	H5				H9
I	Faulted	I1	I2	I3	I4	I5	I6	I7	I8	

The table above is only applicable to ConnectorId > 0. For ConnectorId 0, only a limited set is applicable, namely: Available, Unavailable and Faulted.

The next table describes events that may lead to a status change:

	DESCRIPTION
A2	Usage is initiated (e.g. insert plug, bay occupancy detection, present idTag, push start button, receipt of a RemoteStartTransaction.req)
A3	Can be possible in a Charge Point without an authorization means
A4	Similar to A3 but the EV does not start charging
A5	Similar to A3 but the EVSE does not allow charging
A7	A Reserve Now message is received that reserves the connector
A8	A Change Availability message is received that sets the connector to Unavailable
A9	A fault is detected that prevents further charging operations
B1	Intended usage is ended (e.g. plug removed, bay no longer occupied, second presentation of idTag, time out (configured by the configuration key: `ConnectionTimeOut`) on expected user action)
B3	All prerequisites for charging are met and charging process starts
B4	All prerequisites for charging are met but EV does not start charging
B5	All prerequisites for charging are met but EVSE does not allow charging
B6	Timed out. Usage was initiated (e.g. insert plug, bay occupancy detection), but idTag not presented within timeout.
B9	A fault is detected that prevents further charging operations
C1	Charging session ends while no user action is required (e.g. fixed cable was removed on EV side)
C4	Charging stops upon EV request (e.g. S2 is opened)
C5	Charging stops upon EVSE request (e.g. smart charging restriction, transaction is invalidated by the AuthorizationStatus in a StartTransaction.conf)
C6	Transaction is stopped by user or a Remote Stop Transaction message and further user action is required (e.g. remove cable, leave parking bay)
C8	Charging session ends, no user action is required and the connector is scheduled to become *Unavailable*
C9	A fault is detected that prevents further charging operations
D1	Charging session ends while no user action is required
D3	Charging resumes upon request of the EV (e.g. S2 is closed)
D5	Charging is suspended by EVSE (e.g. due to a smart charging restriction)

	DESCRIPTION
D6	Transaction is stopped and further user action is required
D8	Charging session ends, no user action is required and the connector is scheduled to become *Unavailable*
D9	A fault is detected that prevents further charging operations
E1	Charging session ends while no user action is required
E3	Charging resumes because the EVSE restriction is lifted
E4	The EVSE restriction is lifted but the EV does not start charging
E6	Transaction is stopped and further user action is required
E8	Charging session ends, no user action is required and the connector is scheduled to become *Unavailable*
E9	A fault is detected that prevents further charging operations
F1	All user actions completed
F2	User restart charging session (e.g. reconnects cable, presents idTag again), thereby creating a new Transaction
F8	All user actions completed and the connector is scheduled to become *Unavailable*
F9	A fault is detected that prevents further charging operations
G1	Reservation expires or a Cancel Reservation message is received
G2	Reservation identity is presented
G8	Reservation expires or a Cancel Reservation message is received and the connector is scheduled to become *Unavailable*
G9	A fault is detected that prevents further charging operations
H1	Connector is set *Available* by a Change Availability message
H2	Connector is set *Available* after a user had interacted with the Charge Point
H3	Connector is set *Available* and no user action is required to start charging
H4	Similar to H3 but the EV does not start charging
H5	Similar to H3 but the EVSE does not allow charging
H9	A fault is detected that prevents further charging operations
I1-I8	Fault is resolved and status returns to the pre-fault state

A Charge Point Connector MAY have any of the 9 statuses as shown in the table above. For ConnectorId 0, only a limited set is applicable, namely: Available, Unavailable and Faulted. The status of ConnectorId 0 has no direct connection to the status of the individual Connectors (>0).

If charging is suspended both by the EV and the EVSE, status *SuspendedEVSE* SHALL have precedence over status *SuspendedEV*.

When a Charge Point or a Connector is set to status Unavailable by a Change Availability command, the 'Unavailable' status MUST be persistent across reboots. The Charge Point MAY use the *Unavailable* status internally for other purposes (e.g. while updating firmware or waiting for an initial *Accepted* RegistrationStatus)

As the status *Occupied* has been split into five new statuses (*Preparing*, *Charging*, *SuspendedEV*, *SuspendedEVSE* and *Finishing*), more StatusNotification.req PDUs will be sent from Charge Point to the Central System. For instance, when a transaction is started, the Connector status would successively change from *Preparing* to *Charging* with a short *SuspendedEV* and/or *SuspendedEVSE* inbetween, possibly within a couple of seconds.

To limit the number of transitions, the Charge Point MAY omit sending a StatusNotification.req if it was active for less time than defined in the optional configuration key `MinimumStatusDuration`. This way, a Charge Point MAY choose not to send certain StatusNotification.req PDUs.

A Charge Point manufacturer MAY have implemented a minimal status duration for certain status transitions separate of the `MinimumStatusDuration` setting. The time set in `MinimumStatusDuration` will be added to this default delay. Setting `MinimumStatusDuration` to zero SHALL NOT override the default manufacturer's minimal status duration.

Setting a high `MinimumStatusDuration` time may result in the delayed sending of all StatusNotifications, since the Charge Point will only send the StatusNotification.req once the `MinimumStatusDuration` time is passed.

The Charge Point MAY send a StatusNotification.req PDU to inform the Central System of fault conditions. When the 'status' field is not *Faulted*, the condition should be considered a warning since charging operations are still possible.

ChargePointErrorCode *EVCommunicationError* SHALL only be used with status Preparing, SuspendedEV, SuspendedEVSE and Finishing and be treated as warning.

When a Charge Point is configured with StopTransactionOnEVSideDisconnect set to *false*, a transaction is running and the EV becomes disconnected on EV side, then a StatusNotification.req with the state: *SuspendedEV* SHOULD be send to the Central System, with the 'errorCode' field set to: 'NoError'. The Charge Point SHOULD add additional information in the 'info' field, Notifying the Central System with the reason of suspension: 'EV side disconnected'. The current transaction is not stopped.

When a Charge Point is configured with StopTransactionOnEVSideDisconnect set to *true*, a transaction is running and the EV becomes disconnected on EV side, then a StatusNotification.req with the state: 'Finishing' SHOULD be send to the Central System, with the 'errorCode' field set to: 'NoError'. The Charge Point SHOULD add additional information in the 'info' field, Notifying the Central System with the reason of stopping: 'EV side disconnected'. The current transaction is stopped.

When a Charge Point connects to a Central System after having been offline, it updates the Central System about its status according to the following rules:

1. The Charge Point SHOULD send a StatusNotification.req PDU with its current status if the status changed while the Charge Point was *offline*.
2. The Charge Point MAY send a StatusNotification.req PDU to report an error that occurred while the Charge Point was *offline*.
3. The Charge Point SHOULD NOT send StatusNotification.req PDUs for historical status change events that happened while the Charge Point was offline and that do not inform the Central System of Charge Point errors or the Charge Point's current status.
4. The StatusNotification.req messages MUST be sent in the order in which the events that they describe occurred.

Upon receipt of a StatusNotification.req PDU, the Central System SHALL respond with a StatusNotification.conf PDU.

4.10. Stop Transaction

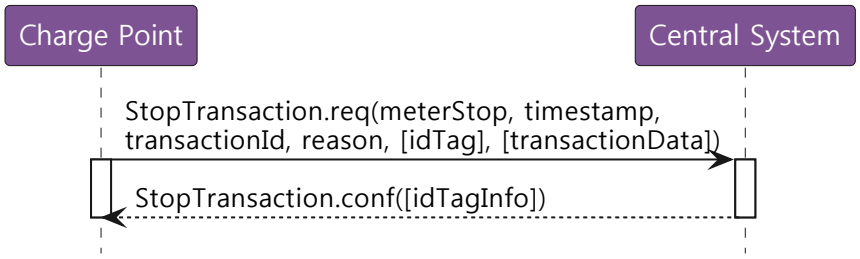

Figure 21. Sequence Diagram: Stop Transaction

When a transaction is stopped, the Charge Point SHALL send a StopTransaction.req PDU, notifying to the Central System that the transaction has stopped.

A StopTransaction.req PDU MAY contain an optional TransactionData element to provide more details about transaction usage. The optional TransactionData element is a container for any number of MeterValues, using the same data structure as the **meterValue** elements of the MeterValues.req PDU (See section MeterValues)

Upon receipt of a StopTransaction.req PDU, the Central System SHALL respond with a StopTransaction.conf PDU.

ChargePointErrorCode *EVCommunicationError* SHALL only be used with status Preparing, SuspendedEV, SuspendedEVSE and Finishing and be treated as warning.

The Central System cannot prevent a transaction from stopping. It MAY only inform the Charge Point it has received the StopTransaction.req and MAY send information about the idTag used to stop the transaction. This information SHOULD be used to update the Authorization Cache if implemented.

The idTag in the request PDU MAY be omitted when the Charge Point itself needs to stop the transaction. For instance, when the Charge Point is requested to reset.

If a transaction is ended in a normal way (e.g. EV-driver presented his identification to stop the transaction), the Reason element MAY be omitted and the Reason SHOULD be assumed 'Local'. If the transaction is not ended normally, the Reason SHOULD be set to a correct value. As part of the normal transaction termination, the Charge Point SHALL unlock the cable (if not permanently attached).

The Charge Point MAY unlock the cable (if not permanently attached) when the cable is disconnected at the EV. If supported, this functionality is reported and controlled by the configuration key `UnlockConnectorOnEVSideDisconnect`

The Charge Point MAY stop a running transaction when the cable is disconnected at the EV. If supported, this functionality is reported and controlled by the configuration key `StopTransactionOnEVSideDisconnect`.

If `StopTransactionOnEVSideDisconnect` is set to *false*, the transaction SHALL not be stopped when the cable is disconnected from the EV. If the EV is reconnected, energy transfer is allowed again. In this case there is no mechanism to prevent other EVs from charging and disconnecting during that same ongoing transaction. With `UnlockConnectorOnEVSideDisconnect` set to *false*, the Connector SHALL remain locked at the Charge Point until the user presents the identifier.

By setting `StopTransactionOnEVSideDisconnect` to *true*, the transaction SHALL be stopped when the cable is disconnected from the EV. If the EV is reconnected, energy transfer is not allowed until the transaction is stopped and a new transaction is started. If `UnlockConnectorOnEVSideDisconnect` is set to *true*, also the Connector on the Charge Point will be unlocked.

If `StopTransactionOnEVSideDisconnect` is set to *false*, this SHALL have priority over `UnlockConnectorOnEVSideDisconnect`. In other words: cables always remain locked when the cable is disconnected at EV side when `StopTransactionOnEVSideDisconnect` is *false*.

Setting `StopTransactionOnEVSideDisconnect` to *true* will prevent sabotage acts to stop the energy flow by unplugging not locked cables on EV side.

It is likely that The Central System applies sanity checks to the data contained in a StopTransaction.req it received. The outcome of such sanity checks SHOULD NOT ever cause the Central System to not respond with a StopTransaction.conf. Failing to respond with a StopTransaction.conf will only cause the Charge Point to try the same message again as specified in Error responses to transaction-related messages.

If Charge Point has implemented an Authorization Cache, then upon receipt of a StopTransaction.conf PDU the Charge Point SHALL update the cache entry, if the idTag is not in the Local Authorization List, with the IdTagInfo value from the response as described under Authorization Cache.

Chapter 5. Operations Initiated by Central System

5.1. Cancel Reservation

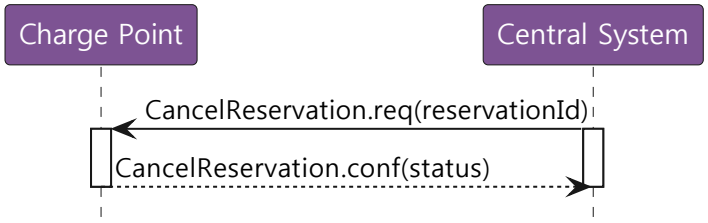

Figure 22. Sequence Diagram: Cancel Reservation

To cancel a reservation the Central System SHALL send an CancelReservation.req PDU to the Charge Point.

If the Charge Point has a reservation matching the reservationId in the request PDU, it SHALL return status 'Accepted'. Otherwise it SHALL return 'Rejected'

5.2. Change Availability

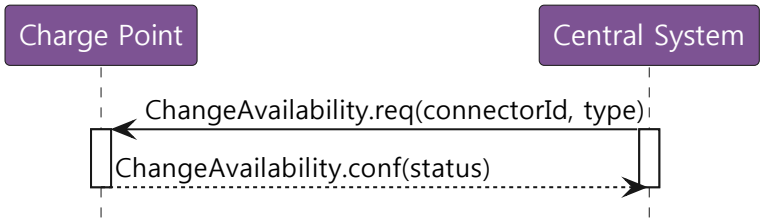

Figure 23. Sequence Diagram: Change Availability

Central System can request a Charge Point to change its availability. A Charge Point is considered available ("operative") when it is charging or ready for charging. A Charge Point is considered unavailable when it does not allow any charging. The Central System SHALL send a ChangeAvailability.req PDU for requesting a Charge Point to change its availability. The Central System can change the availability to available or unavailable.

Upon receipt of a ChangeAvailability.req PDU, the Charge Point SHALL respond with a ChangeAvailability.conf PDU. The response PDU SHALL indicate whether the Charge Point is able to change to the requested availability or not. When a transaction is in progress Charge Point SHALL respond with availability status 'Scheduled' to indicate that it is scheduled to occur after the transaction has finished.

In the event that Central System requests Charge Point to change to a status it is already in, Charge Point SHALL respond with availability status 'Accepted'.

When an availability change requested with a ChangeAvailability.req PDU has happened, the Charge Point SHALL inform Central System of its new availability status with a StatusNotification.req as described there.

In the case the ChangeAvailability.req contains ConnectorId = 0, the status change applies to the Charge Point and all Connectors.

Persistent states: for example: Connector set to Unavailable shall persist a reboot.

5.3. Change Configuration

Figure 24. Sequence Diagram: Change Configuration

Central System can request a Charge Point to change configuration parameters. To achieve this, Central System SHALL send a ChangeConfiguration.req. This request contains a key-value pair, where "key" is the name of the configuration setting to change and "value" contains the new setting for the configuration setting.

Upon receipt of a ChangeConfiguration.req Charge Point SHALL reply with a ChangeConfiguration.conf indicating whether it was able to apply the change to its configuration. Content of "key" and "value" is not prescribed. The Charge Point SHALL set the status field in the ChangeConfiguration.conf according to the following rules:

- If the change was applied successfully, and the change if effective immediately, the Charge Point SHALL respond with a status 'Accepted'.
- If the change was applied successfully, but a reboot is needed to make it effective, the Charge Point SHALL respond with status 'RebootRequired'.
- If "key" does not correspond to a configuration setting supported by Charge Point, it SHALL respond with a status 'NotSupported'.
- If the Charge Point did not set the configuration, and none of the previous statuses applies, the Charge Point SHALL respond with status 'Rejected'.

Examples of Change Configuration requests to which a Charge Point responds with a ChangeConfiguration.conf with a status of 'Rejected' are requests with out-of-range values and requests with values that do not conform to an expected format.

If a key value is defined as a CSL, it MAY be accompanied with a `[KeyName]MaxLength` key, indicating the max length of the CSL in items. If this key is not set, a safe value of 1 (one) item SHOULD be assumed.

5.4. Clear Cache

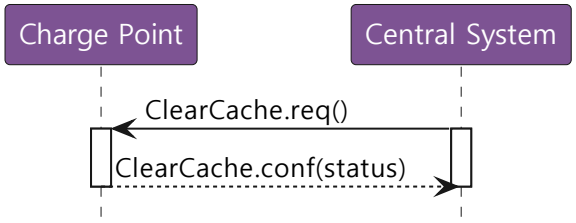

Figure 25. Sequence Diagram: Clear Cache

Central System can request a Charge Point to clear its Authorization Cache. The Central System SHALL send a ClearCache.req PDU for clearing the Charge Point's Authorization Cache.

Upon receipt of a ClearCache.req PDU, the Charge Point SHALL respond with a ClearCache.conf PDU. The response PDU SHALL indicate whether the Charge Point was able to clear its Authorization Cache.

5.5. Clear Charging Profile

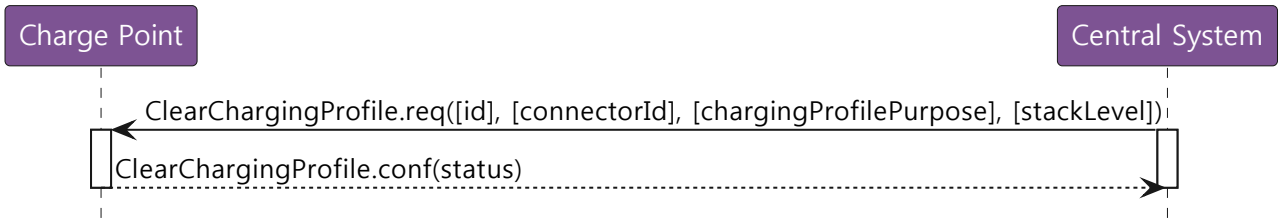

Figure 26. Sequence Diagram: Clear Charging Profile

If the Central System wishes to clear some or all of the charging profiles that were previously sent the Charge Point, it SHALL use the ClearChargingProfile.req PDU.

The Charge Point SHALL respond with a ClearChargingProfile.conf PDU specifying whether it was able to process the request.

5.6. Data Transfer

Figure 27. Sequence Diagram: Data Transfer

If the Central System needs to send information to a Charge Point for a function not supported by OCPP, it SHALL use the DataTransfer.req PDU.

Behaviour of this operation is identical to the Data Transfer operation initiated by the Charge Point. See Data Transfer for details.

5.7. Get Composite Schedule

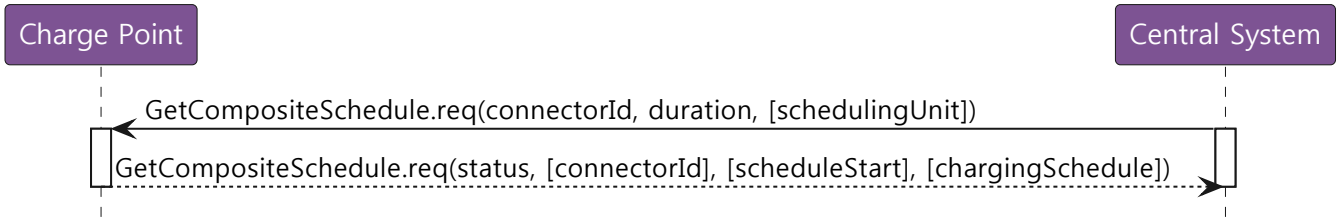

Figure 28. Sequence Diagram: Get Composite Schedule

The Central System MAY request the Charge Point to report the Composite Charging Schedule by sending a GetCompositeSchedule.req PDU. The reported schedule, in the GetCompositeSchedule.conf PDU, is the result of the calculation of all active schedules and possible local limits present in the Charge Point. Local Limits might be taken into account.

Upon receipt of a GetCompositeSchedule.req, the Charge Point SHALL calculate the Composite Charging Schedule intervals, from the moment the request PDU is received: Time X, up to X + Duration, and send them in the GetCompositeSchedule.conf PDU to the Central System.

If the ConnectorId in the request is set to '0', the Charge Point SHALL report the total expected power or current the Charge Point expects to consume from the grid during the requested time period.

Please note that the charging schedule sent by the charge point is only indicative for that point in time. this schedule might change over time due to external causes (for instance, local balancing based on grid connection capacity is active and one Connector becomes available).

If the Charge Point is not able to report the requested schedule, for instance if the connectorId is unknown, it SHALL respond with a status Rejected.

5.8. Get Configuration

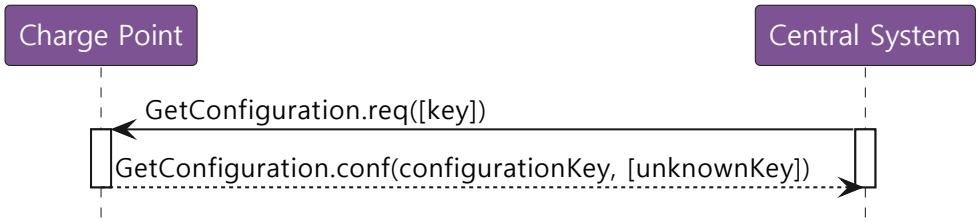

Figure 29. Sequence Diagram: Get Configuration

To retrieve the value of configuration settings, the Central System SHALL send a GetConfiguration.req PDU to the Charge Point.

If the list of keys in the request PDU is empty or missing (it is optional), the Charge Point SHALL return a list of all configuration settings in GetConfiguration.conf. Otherwise Charge Point SHALL return a list of recognized keys and their corresponding values and read-only state. Unrecognized keys SHALL be placed in the response PDU as part of the optional unknown key list element of GetConfiguration.conf.

The number of configuration keys requested in a single PDU MAY be limited by the Charge Point. This maximum can be retrieved by reading the configuration key GetConfigurationMaxKeys.

5.9. Get Diagnostics

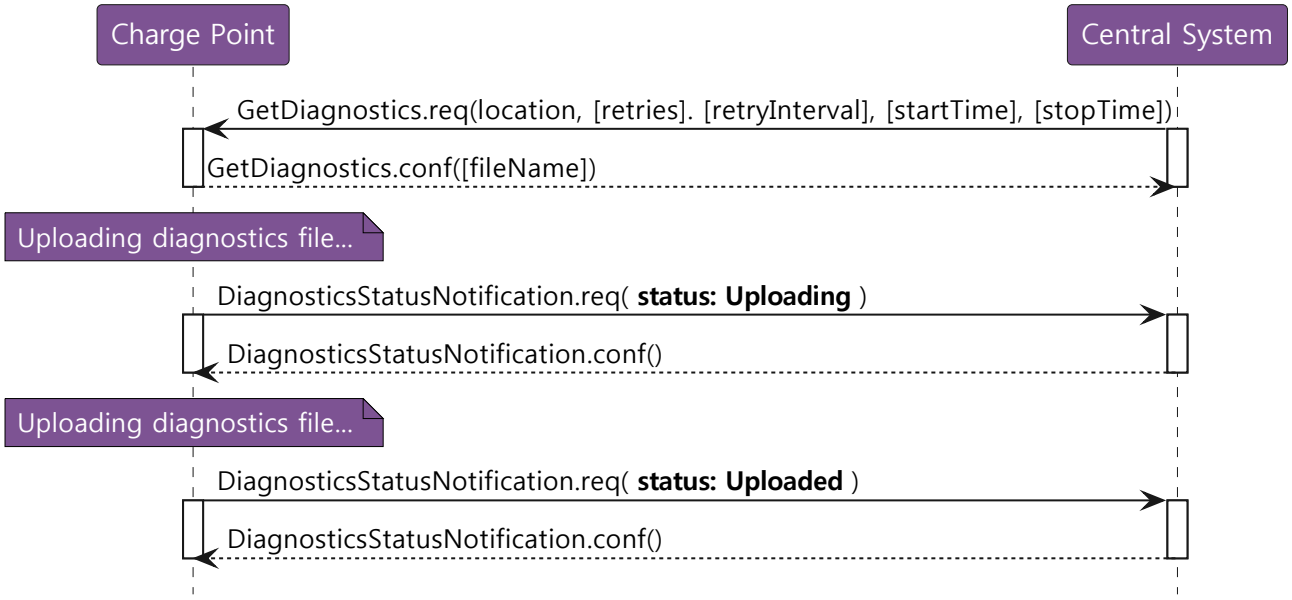

Figure 30. Sequence Diagram: Get Diagnostics

Central System can request a Charge Point for diagnostic information. The Central System SHALL send a GetDiagnostics.req PDU for getting diagnostic information of a Charge Point with a location where the Charge Point MUST upload its diagnostic data to and optionally a begin and end time for the requested diagnostic information.

Upon receipt of a GetDiagnostics.req PDU, and if diagnostics information is available then Charge Point SHALL respond with a GetDiagnostics.conf PDU stating the name of the file containing the diagnostic information that will be uploaded. Charge Point SHALL upload a single file. Format of the diagnostics file is not prescribed. If no diagnostics file is available, then GetDiagnostics.conf SHALL NOT contain a file name.

During uploading of a diagnostics file, the Charge Point MUST send DiagnosticsStatusNotification.req PDUs to keep the Central System updated with the status of the upload process.

5.10. Get Local List Version

Figure 31. Sequence Diagram: Get Local List Version

In order to support synchronisation of the Local Authorization List, Central System can request a Charge Point for the version number of the Local Authorization List. The Central System SHALL send a GetLocalListVersion.req PDU to request this value.

Upon receipt of a GetLocalListVersion.req PDU Charge Point SHALL respond with a GetLocalListVersion.conf PDU containing the version number of its Local Authorization List. A version number of 0 (zero) SHALL be used to indicate that the local authorization list is empty, and a version number of -1 SHALL be used to indicate that the Charge Point does not support Local Authorization Lists.

5.11. Remote Start Transaction

Figure 32. Sequence Diagram: Remote Start Transaction

Central System can request a Charge Point to start a transaction by sending a RemoteStartTransaction.req. Upon receipt, the Charge Point SHALL reply with RemoteStartTransaction.conf and a status indicating whether it has accepted the request and will attempt to start a transaction.

The effect of the RemoteStartTransaction.req message depends on the value of the AuthorizeRemoteTxRequests configuration key in the Charge Point.

- If the value of `AuthorizeRemoteTxRequests` is *true*, the Charge Point SHALL behave as if in response to a local action at the Charge Point to start a transaction with the idTag given in the RemoteStartTransaction.req message. This means that the Charge Point will first try to authorize the idTag, using the Local Authorization List, Authorization Cache and/or an Authorize.req request. A transaction will only be started after authorization was obtained.
- If the value of `AuthorizeRemoteTxRequests` is *false*, the Charge Point SHALL immediately try to start a transaction for the idTag given in the RemoteStartTransaction.req message. Note that after the transaction has been started, the Charge Point will send a StartTransaction request to the Central System, and the Central System will check the authorization status of the idTag when processing this StartTransaction request.

The following typical use cases are the reason for Remote Start Transaction:

- Enable a CPO operator to help an EV driver that has problems starting a transaction.
- Enable mobile apps to control charging transactions via the Central System.
- Enable the use of SMS to control charging transactions via the Central System.

The RemoteStartTransaction.req SHALL contain an identifier (idTag), which Charge Point SHALL use, if it is able to start a transaction, to send a StartTransaction.req to Central System. The transaction is started in the same way as described in StartTransaction. The RemoteStartTransaction.req MAY contain a connector id if the transaction is to be started on a specific connector. When no connector id is provided, the Charge Point is in control of the connector selection. A Charge Point MAY reject a RemoteStartTransaction.req without a connector id.

The Central System MAY include a ChargingProfile in the RemoteStartTransaction request. The purpose of this ChargingProfile SHALL be set to TxProfile. If accepted, the Charge Point SHALL use this ChargingProfile for the transaction.

If a Charge Point without support for Smart Charging receives a RemoteStartTransaction.req with a Charging Profile, this parameter SHOULD be ignored.

5.12. Remote Stop Transaction

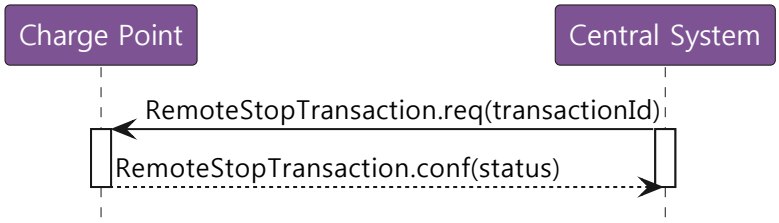

Figure 33. Remote Stop Transaction

Central System can request a Charge Point to stop a transaction by sending a RemoteStopTransaction.req to Charge Point with the identifier of the transaction. Charge Point SHALL reply with RemoteStopTransaction.conf and a status indicating whether it has accepted the request and a transaction with the given transactionId is ongoing and will be stopped.

This remote request to stop a transaction is equal to a local action to stop a transaction. Therefore, the transaction SHALL be stopped, The Charge Point SHALL send a StopTransaction.req and, if applicable, unlock the connector.

The following two main use cases are the reason for Remote Stop Transaction:

- Enable a CPO operator to help an EV driver that has problems stopping a transaction.
- Enable mobile apps to control charging transactions via the Central System.

5.13. Reserve Now

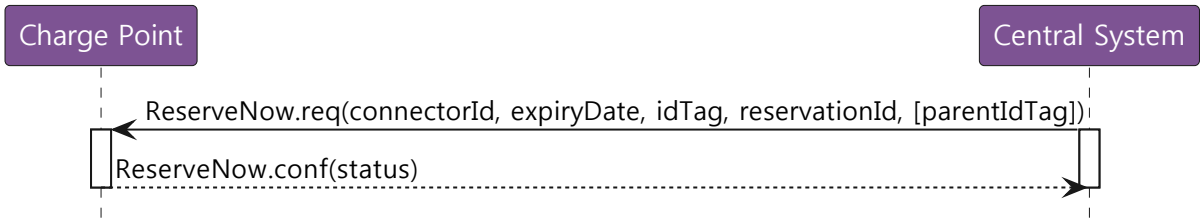

Figure 34. Sequence Diagram: Reserve Now

A Central System can issue a ReserveNow.req to a Charge Point to reserve a connector for use by a specific idTag.

To request a reservation the Central System SHALL send a ReserveNow.req PDU to a Charge Point. The Central System MAY specify a connector to be reserved. Upon receipt of a ReserveNow.req PDU, the Charge Point SHALL respond with a ReserveNow.conf PDU.

If the reservationId in the request matches a reservation in the Charge Point, then the Charge Point SHALL replace that reservation with the new reservation in the request.

If the reservationId does not match any reservation in the Charge Point, then the Charge Point SHALL return the status value 'Accepted' if it succeeds in reserving a connector. The Charge Point SHALL return 'Occupied' if the Charge Point or the specified connector are occupied. The Charge Point SHALL also return 'Occupied' when the Charge Point or connector has been reserved for the same or another idTag. The Charge Point SHALL return 'Faulted' if the Charge Point or the connector are in the Faulted state. The Charge Point SHALL return 'Unavailable' if the Charge Point or connector are in the Unavailable state. The Charge Point SHALL return 'Rejected' if it is configured not to accept reservations.

If the Charge Point accepts the reservation request, then it SHALL refuse charging for all incoming idTags on the reserved connector,

except when the incoming idTag or the parent idTag match the idTag or parent idTag of the reservation.

When the configuration key: `ReserveConnectorZeroSupported` is set to *true* the Charge Point supports reservations on connector 0. If the connectorId in the reservation request is 0, then the Charge Point SHALL NOT reserve a specific connector, but SHALL make sure that at any time during the validity of the reservation, one connector remains available for the reserved idTag. If the configuration key: `ReserveConnectorZeroSupported` is not set or set to *false*, the Charge Point SHALL return 'Rejected'.

If the parent idTag in the reservation has a value (it is optional), then in order to determine the parent idTag that is associated with an incoming idTag, the Charge Point MAY look it up in its Local Authorization List or Authorization Cache. If it is not found in the Local Authorization List or Authorization Cache, then the Charge Point SHALL send an Authorize.req for the incoming idTag to the Central System. The Authorize.conf response MAY contain the parent-id.

A reservation SHALL be terminated on the Charge Point when either (1) a transaction is started for the reserved idTag or parent idTag and on the reserved connector or any connector when the reserved connectorId is 0, or (2) when the time specified in expiryDate is reached, or (3) when the Charge Point or connector are set to Faulted or Unavailable.

If a transaction for the reserved idTag is started, then Charge Point SHALL send the reservationId in the StartTransaction.req PDU (see Start Transaction) to notify the Central System that the reservation is terminated.

When a reservation expires, the Charge Point SHALL terminate the reservation and make the connector available. The Charge Point SHALL send a status notification to notify the Central System that the reserved connector is now available.

If Charge Point has implemented an Authorization Cache, then upon receipt of a ReserveNow.conf PDU the Charge Point SHALL update the cache entry, if the idTag is not in the Local Authorization List, with the IdTagInfo value from the response as described under Authorization Cache.

It is RECOMMENDED to validate the Identifier with an authorize.req after reception of a ReserveNow.req and before the start of the transaction.

5.14. Reset

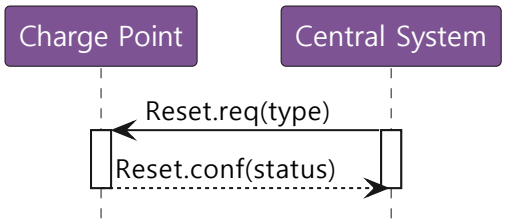

Figure 35. Sequence Diagram: Reset

The Central System SHALL send a Reset.req PDU for requesting a Charge Point to reset itself. The Central System can request a hard or a soft reset. Upon receipt of a Reset.req PDU, the Charge Point SHALL respond with a Reset.conf PDU. The response PDU SHALL include whether the Charge Point will attempt to reset itself.

After receipt of a Reset.req, The Charge Point SHALL send a StopTransaction.req for any ongoing transaction before performing the reset. If the Charge Point fails to receive a StopTransaction.conf form the Central System, it shall queue the StopTransaction.req

At receipt of a soft reset, the Charge Point SHALL stop ongoing transactions gracefully and send StopTransaction.req for every ongoing transaction. It should then restart the application software (if possible otherwise restart the processor/controller).

At receipt of a hard reset the Charge Point SHALL restart (all) the hardware, it is not required to gracefully stop ongoing transaction. If possible the Charge Point sends a StopTransaction.req for previously ongoing transactions after having restarted and having been accepted by the Central System via a BootNotification.conf. This is a last resort solution for a not correctly functioning Charge Points, by sending a "hard" reset, (queued) information might get lost

Persistent states: for example: Connector set to Unavailable shall persist

5.15. Send Local List

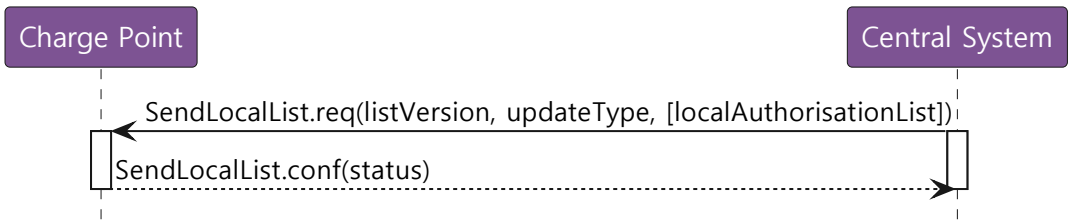

Figure 36. Sequence Diagram: Send Local List

Central System can send a Local Authorization List that a Charge Point can use for authorization of idTags. The list MAY be either a full list to replace the current list in the Charge Point or it MAY be a differential list with updates to be applied to the current list in the Charge Point.

The Central System SHALL send a SendLocalList.req PDU to send the list to a Charge Point. The SendLocalList.req PDU SHALL contain the type of update (full or differential) and the version number that the Charge Point MUST associate with the local authorization list after it has been updated.

Upon receipt of a SendLocalList.req PDU, the Charge Point SHALL respond with a SendLocalList.conf PDU. The response PDU SHALL indicate whether the Charge Point has accepted the update of the local authorization list. If the status is Failed or VersionMismatch and the updateType was Differential, then Central System SHOULD retry sending the full local authorization list with updateType Full.

5.16. Set Charging Profile

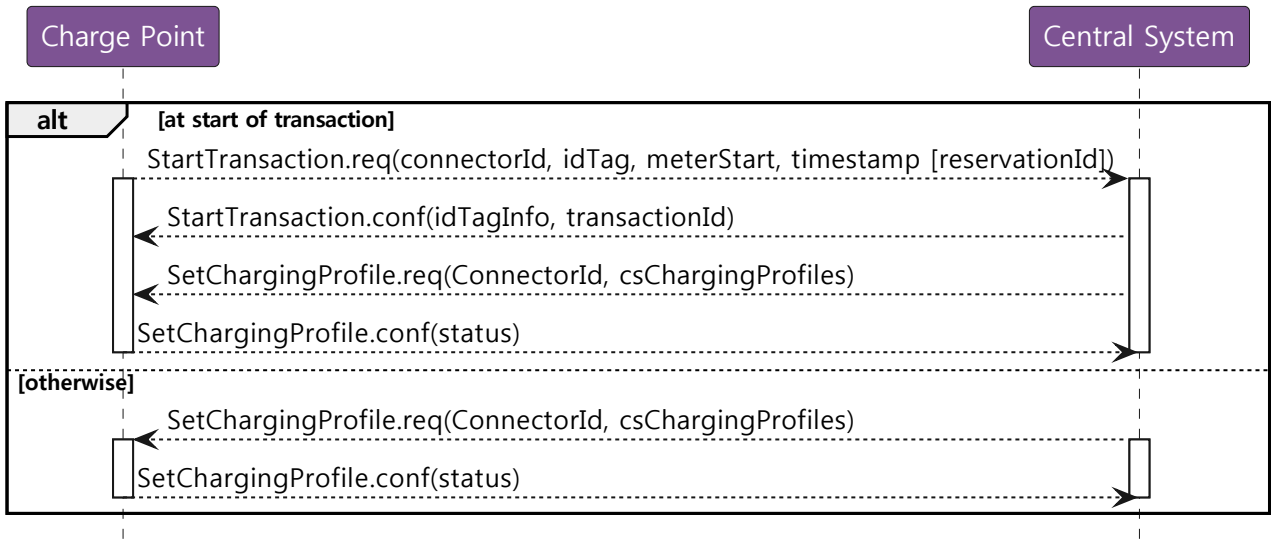

Figure 37. Sequence Diagram: Set Charging Profile

A Central System can send a SetChargingProfile.req to a Charge Point, to set a charging profile, in the following situations:

- At the start of a transaction to set the charging profile for the transaction;
- In a RemoteStartTransaction.req sent to a Charge Point
- During a transaction to change the active profile for the transaction;
- Outside the context of a transaction as a separate message to set a charging profile to a local controller, Charge Point, or a default charging profile to a connector.

To prevent mismatch between transactions and a TxProfile, The Central System SHALL include the transactionId in a SetChargingProfile.req if the profile applies to a specific transaction.

These situations are described below.

5.16.1. Setting a charging profile at start of transaction

If the Central System receives a StartTransaction.req the Central System SHALL respond with a StartTransaction.conf. If there is a need for a charging profile, The Central System MAY choose to send a SetChargingProfile.req to the Charge Point.

It is RECOMMENDED to check the timestamp in the StartTransaction.req PDU prior to sending a charging profile to check if the transaction is likely to be still ongoing. The StartTransaction.req might have been cached during an offline period.

5.16.2. Setting a charge profile in a RemoteStartTransaction request

The Central System MAY include a charging profile in a RemoteStartTransaction request.

If the Central System includes a ChargingProfile, the ChargingProfilePurpose MUST be set to TxProfile and the transactionId SHALL NOT be set.

The Charge Point SHALL apply the given profile to the newly started transaction. This transaction will get a transactionId assigned by Central System via a StartTransaction.conf. When the Charge Point receives a SetChargingProfile.req, with the *transactionId* for this transaction, with the same StackLevel as the profile given in the RemoteStartTransaction.req, the Charge Point SHALL replace the existing charging profile, otherwise it SHALL install/stack the profile next to the already existing profile(s).

5.16.3. Setting a charging profile during a transaction

The Central System MAY send a charging profile to a Charge Point to update the charging profile for that transaction. The Central System SHALL use the SetChargingProfile.req PDU for that purpose. If a charging profile with the same chargingProfileId, or the same combination of stackLevel / ChargingProfilePurpose, exists on the Charge Point, the new charging profile SHALL replace the existing charging profile, otherwise it SHALL be added. The Charge Point SHALL then re-evaluate its collection of charge profiles to determine which charging profile will become active. In order to ensure that the updated charging profile applies only to the current transaction, the chargingProfilePurpose of the ChargingProfile MUST be set to TxProfile. (See section: Charging Profile Purposes)

5.16.4. Setting a charging profile outside of a transaction

The Central System MAY send charging profiles to a Charge Point that are to be used as default charging profiles The Central System SHALL use the SetChargingProfile.req PDU for that purpose. Such charging profiles MAY be sent at any time. If a charging profile with the same chargingProfileId, or the same combination of stackLevel / ChargingProfilePurpose, exists on the Charge Point, the new charging profile SHALL replace the existing charging profile, otherwise it SHALL be added. The Charge Point SHALL then re-evaluate its collection of charge profiles to determine which charging profile will become active.

It is not possible to set a ChargingProfile with purpose set to TxProfile without presence of an active transaction, or in advance of a transaction.

When a ChargingProfile is refreshed during execution, it is advised to put the startSchedule of the new ChargingProfile in the past, so there is no period of default charging behaviour inbetween the ChargingProfiles. The Charge Point SHALL continue to execute the existing ChargingProfile until the new ChargingProfile is installed.

If the chargingSchedulePeriod is longer than duration , the remainder periods SHALL not be executed. If duration is longer than the chargingSchedulePeriod, the Charge Point SHALL keep the value of the last chargingSchedulePeriod until *duration* has ended.

When recurrencyKind is used in combination with a chargingSchedulePeriod and/or duration that is longer then the recurrence period duration, the remainder periods SHALL not be executed.

The StartSchedule of the first chargingSchedulePeriod in a chargingSchedule SHALL always be 0.

When recurrencyKind is used in combination with a chargingSchedule *duration* shorter than the recurrencyKind period, the Charge Point SHALL fall back to default behaviour after the chargingSchedule *duration* ends. This fall back means that the Charge Point SHALL use a ChargingProfile with a lower stackLevel if available. If no other ChargingProfile is available, the Charge Point SHALL allow charging as if no ChargingProfile is installed. If the chargingSchedulePeriod and/or duration is longer then the recurrence period duration, the remainder periods SHALL not be executed.

5.17. Trigger Message

Figure 38. Sequence Diagram: Trigger Message

During normal operation, the Charge Point informs the Central System of its state and any relevant occurrences. If there is nothing to report the Charge Point will send at least a heartBeat at a predefined interval. Under normal circumstances this is just fine, but what if the Central System has (whatever) reason to doubt the last known state? What can a Central System do if a firmware update is in progress and the last status notification it received about it was much longer ago than could reasonably be expected? The same can be asked for the progress of a diagnostics request. The problem in these situations is not that the information needed isn't covered by existing messages, the problem is strictly a timing issue. The Charge Point has the information, but has no way of knowing that the Central System would like an update.

The TriggerMessage.req makes it possible for the Central System, to request the Charge Point, to send Charge Point-initiated messages. In the request the Central System indicates which message it wishes to receive. For every such requested message the Central System MAY optionally indicate to which connector this request applies. The requested message is leading: if the specified connectorId is not relevant to the message, it should be ignored. In such cases the requested message should still be sent.

Inversely, if the connectorId is relevant but absent, this should be interpreted as "for all allowed connectorId values". For example, a request for a statusNotification for connectorId 0 is a request for the status of the Charge Point. A request for a statusNotification without connectorId is a request for multiple statusNotifications: the notification for the Charge Point itself and a notification for each of its connectors.

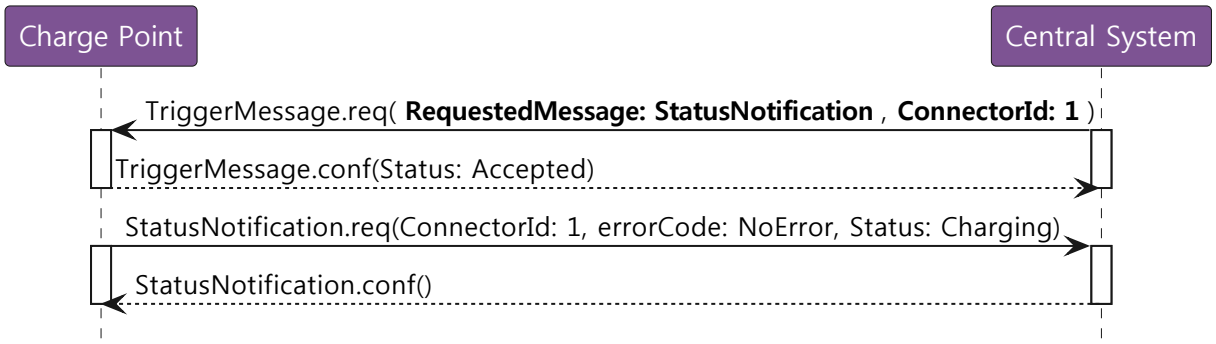

Figure 39. Sequence Diagram: Trigger Message StatusNotification Example

The Charge Point SHALL first send the TriggerMessage response, before sending the requested message. In the TriggerMessage.conf the Charge Point SHALL indicate whether it will send it or not, by returning ACCEPTED or REJECTED. It is up to the Charge Point if it

accepts or rejects the request to send. If the requested message is unknown or not implemented the Charge Point SHALL return NOT_IMPLEMENTED.

Messages that the Charge Point marks as accepted SHOULD be sent. The situation could occur that, between accepting the request and actually sending the requested message, that same message gets sent because of normal operations. In such cases the message just sent MAY be considered as complying with the request.

The TriggerMessage mechanism is not intended to retrieve historic data. The messages it triggers should only give current information. A MeterValues.req message triggered in this way for instance SHALL return the most recent measurements for all measurands configured in configuration key `MeterValuesSampledData`. StartTransaction and StopTransaction have been left out of this mechanism because they are not state related, but by their nature describe a transition.

5.18. Unlock Connector

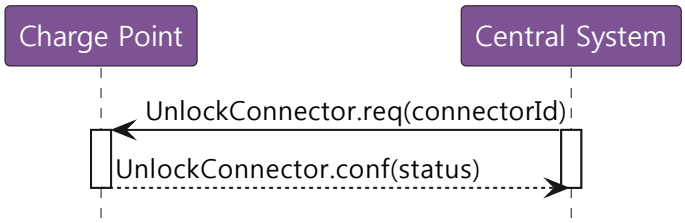

Figure 40. Sequence Diagram: Unlock Connector

Central System can request a Charge Point to unlock a connector. To do so, the Central System SHALL send an UnlockConnector.req PDU. The purpose of this message: Help EV drivers that have problems unplugging their cable from the Charge Point

in case of malfunction of the Connector cable retention. When a EV driver calls the CPO help-desk, an operator could manually trigger the sending of an UnlockConnector.req to the Charge Point, forcing a new attempt to unlock the connector. Hopefully this time the connector unlocks and the EV driver can unplug the cable and drive away.

The UnlockConnector.req SHOULD NOT be used to remotely stop a running transaction, use the Remote Stop Transaction instead.

Upon receipt of an UnlockConnector.req PDU, the Charge Point SHALL respond with a UnlockConnector.conf PDU. The response PDU SHALL indicate whether the Charge Point was able to unlock its connector.

If there was a transaction in progress on the specific connector, then Charge Point SHALL finish the transaction first as described in Stop Transaction.

UnlockConnector.req is intented only for unlocking the cable retention lock on the Connector not for unlocking a connector access door.

5.19. Update Firmware

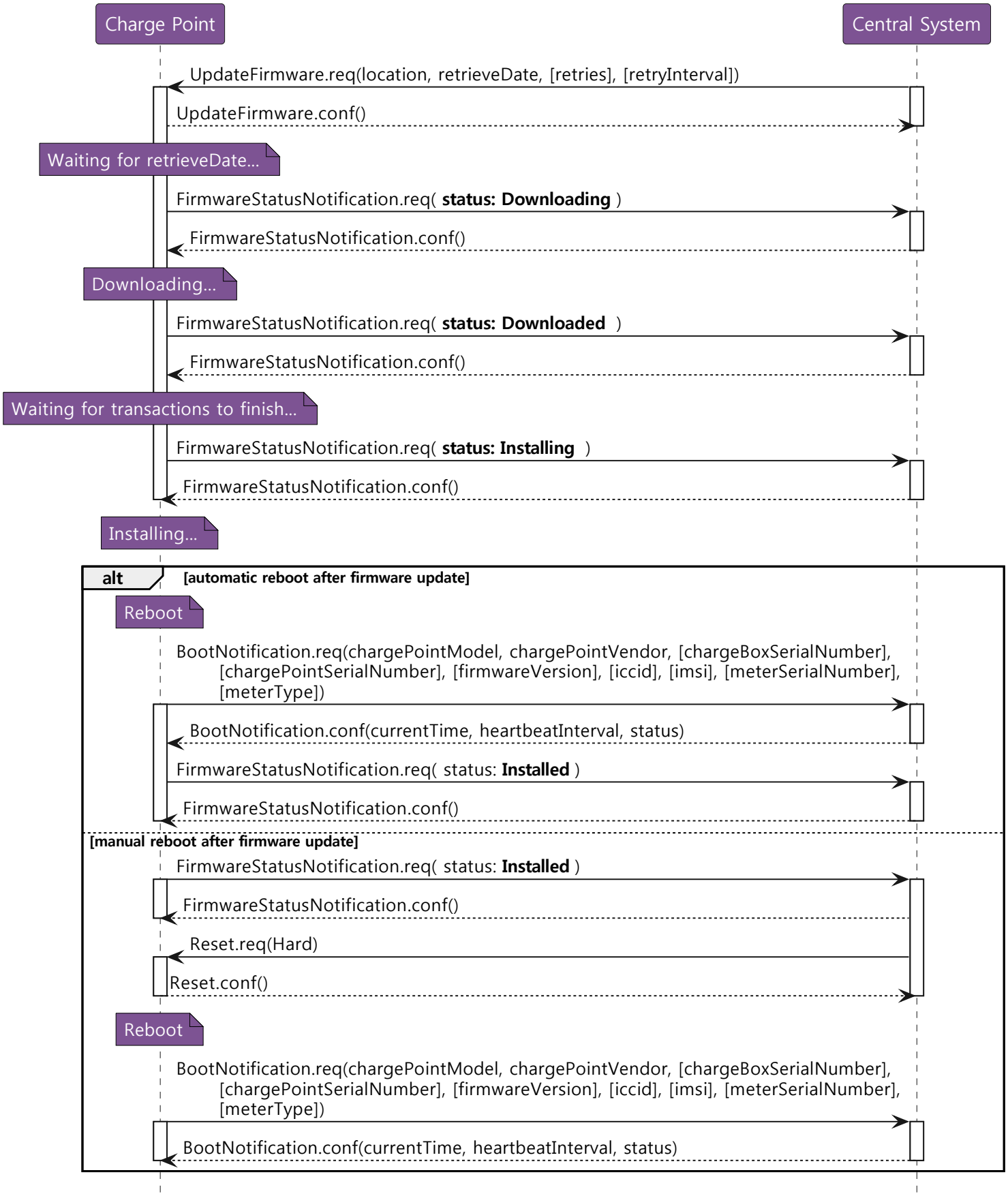

Figure 41. Sequence Diagram: Update Firmware

Central System can notify a Charge Point that it needs to update its firmware. The Central System SHALL send an UpdateFirmware.req PDU to instruct the Charge Point to install new firmware. The PDU SHALL contain a date and time after which the Charge Point is allowed to retrieve the new firmware and the location from which the firmware can be downloaded.

Upon receipt of an UpdateFirmware.req PDU, the Charge Point SHALL respond with a UpdateFirmware.conf PDU. The Charge Point SHOULD start retrieving the firmware as soon as possible after retrieve-date.

During downloading and installation of the firmware, the Charge Point MUST send FirmwareStatusNotification.req PDUs to keep the Central System updated with the status of the update process.

The Charge Point SHALL, if the new firmware image is "valid", install the new firmware as soon as it is able to.

If it is not possible to continue charging during installation of firmware, it is RECOMMENDED to wait until Charging Session has ended (Charge Point idle) before commencing installation. It is RECOMMENDED to set connectors that are not in use to UNAVAILABLE while the Charge Point waits for the Session to end.

The sequence diagram above is an example. It is good practice to first reboot the Charge Point to check the new firmware is booting and able to connect to the Central System, before sending the status: *Installed*. This is not a requirement.

Chapter 6. Messages

6.1. Authorize.req

This contains the field definition of the Authorize.req PDU sent by the Charge Point to the Central System. See also Authorize.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
idTag	IdToken	1..1	Required. This contains the identifier that needs to be authorized.

6.2. Authorize.conf

This contains the field definition of the Authorize.conf PDU sent by the Central System to the Charge Point in response to a Authorize.req PDU. See also Authorize.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
idTagInfo	IdTagInfo	1..1	Required. This contains information about authorization status, expiry and parent id.

6.3. BootNotification.req

This contains the field definition of the BootNotification.req PDU sent by the Charge Point to the Central System. See also Boot Notification.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
chargeBoxSerialNumber	CiString25Type	0..1	Optional. This contains a value that identifies the serial number of the Charge Box inside the Charge Point. Deprecated, will be removed in future version
chargePointModel	CiString20Type	1..1	Required. This contains a value that identifies the model of the ChargePoint.
chargePointSerialNumber	CiString25Type	0..1	Optional. This contains a value that identifies the serial number of the Charge Point.
chargePointVendor	CiString20Type	1..1	Required. This contains a value that identifies the vendor of the ChargePoint.
firmwareVersion	CiString50Type	0..1	Optional. This contains the firmware version of the Charge Point.
iccid	CiString20Type	0..1	Optional. This contains the ICCID of the modem's SIM card.
imsi	CiString20Type	0..1	Optional. This contains the IMSI of the modem's SIM card.
meterSerialNumber	CiString25Type	0..1	Optional. This contains the serial number of the main electrical meter of the Charge Point.
meterType	CiString25Type	0..1	Optional. This contains the type of the main electrical meter of the Charge Point.

6.4. BootNotification.conf

This contains the field definition of the BootNotification.conf PDU sent by the Central System to the Charge Point in response to a BootNotification.req PDU. See also Boot Notification

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
currentTime	dateTime	1..1	Required. This contains the Central System's current time.
interval	integer	1..1	Required. When RegistrationStatus is *Accepted*, this contains the heartbeat interval in seconds. If the Central System returns something other than Accepted, the value of the interval field indicates the minimum wait time before sending a next BootNotification request.
status	RegistrationStatus	1..1	Required. This contains whether the Charge Point has been registered within the System Central.

6.5. CancelReservation.req

This contains the field definition of the CancelReservation.req PDU sent by the Central System to the Charge Point. See also Cancel Reservation.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
reservationId	integer	1..1	Required. Id of the reservation to cancel.

6.6. CancelReservation.conf

This contains the field definition of the CancelReservation.conf PDU sent by the Charge Point to the Central System in response to a CancelReservation.req PDU. See also Cancel Reservation.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	CancelReservationStatus	1..1	Required. This indicates the success or failure of the cancelling of a reservation by Central System.

6.7. ChangeAvailability.req

This contains the field definition of the ChangeAvailability.req PDU sent by the Central System to the Charge Point. See also Change Availability.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer connectorId >= 0	1..1	Required. The id of the connector for which availability needs to change. Id '0' (zero) is used if the availability of the Charge Point and all its connectors needs to change.
type	AvailabilityType	1..1	Required. This contains the type of availability change that the Charge Point should perform.

6.8. ChangeAvailability.conf

This contains the field definition of the ChangeAvailability.conf PDU return by Charge Point to Central System. See also Change Availability.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	AvailabilityStatus	1..1	Required. This indicates whether the Charge Point is able to perform the availability change.

6.9. ChangeConfiguration.req

This contains the field definition of the ChangeConfiguration.req PDU sent by Central System to Charge Point. It is RECOMMENDED that the content and meaning of the 'key' and 'value' fields is agreed upon between Charge Point and Central System. See also Change Configuration.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
key	CiString50Type	1..1	Required. The name of the configuration setting to change. See for standard configuration key names and associated values.
value	CiString500Type	1..1	Required. The new value as string for the setting. See for standard configuration key names and associated values

6.10. ChangeConfiguration.conf

This contains the field definition of the ChangeConfiguration.conf PDU returned from Charge Point to Central System. See also Change Configuration.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	ConfigurationStatus	1..1	Required. Returns whether configuration change has been accepted.

6.11. ClearCache.req

This contains the field definition of the ClearCache.req PDU sent by the Central System to the Charge Point. See also Clear Cache.

No fields are defined.

6.12. ClearCache.conf

This contains the field definition of the ClearCache.conf PDU sent by the Charge Point to the Central System in response to a ClearCache.req PDU. See also Clear Cache.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	ClearCacheStatus	1..1	Required. Accepted if the Charge Point has executed the request, otherwise rejected.

6.13. ClearChargingProfile.req

This contains the field definition of the ClearChargingProfile.req PDU sent by the Central System to the Charge Point.

The Central System can use this message to clear (remove) either a specific charging profile (denoted by id) or a selection of charging profiles that match with the values of the optional connectorId, stackLevel and chargingProfilePurpose fields. See also Clear Charging Profile.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
id	integer	0..1	Optional. The ID of the charging profile to clear.
connectorId	integer	0..1	Optional. Specifies the ID of the connector for which to clear charging profiles. A connectorId of zero (0) specifies the charging profile for the overall Charge Point. Absence of this parameter means the clearing applies to all charging profiles that match the other criteria in the request.
chargingProfilePurpose	ChargingProfilePurposeType	0..1	Optional. Specifies to purpose of the charging profiles that will be cleared, if they meet the other criteria in the request.
stackLevel	integer	0..1	Optional. specifies the stackLevel for which charging profiles will be cleared, if they meet the other criteria in the request.

6.14. ClearChargingProfile.conf

This contains the field definition of the ClearChargingProfile.conf PDU sent by the Charge Point to the Central System in response to a ClearChargingProfile.req PDU. See also Clear Charging Profile.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	ClearChargingProfileStatus	1..1	Required. Indicates if the Charge Point was able to execute the request.

6.15. DataTransfer.req

This contains the field definition of the DataTransfer.req PDU sent either by the Central System to the Charge Point or vice versa. See also Data Transfer.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
vendorId	CiString255Type	1..1	Required. This identifies the Vendor specific implementation
messageId	CiString50Type	0..1	Optional. Additional identification field
data	Text Length undefined	0..1	Optional. Data without specified length or format.

6.16. DataTransfer.conf

This contains the field definition of the DataTransfer.conf PDU sent by the Charge Point to the Central System or vice versa in response to a DataTransfer.req PDU. See also Data Transfer.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	DataTransferStatus	1..1	Required. This indicates the success or failure of the data transfer.
data	Text Length undefined	0..1	Optional. Data in response to request.

6.17. DiagnosticsStatusNotification.req

This contains the field definition of the DiagnosticsStatusNotification.req PDU sent by the Charge Point to the Central System. See also Diagnostics Status Notification.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	DiagnosticsStatus	1..1	Required. This contains the status of the diagnostics upload.

6.18. DiagnosticsStatusNotification.conf

This contains the field definition of the DiagnosticsStatusNotification.conf PDU sent by the Central System to the Charge Point in response to a DiagnosticsStatusNotification.req PDU. See also Diagnostics Status Notification.

No fields are defined.

6.19. FirmwareStatusNotification.req

This contains the field definition of the FirmwareStatusNotifitacion.req PDU sent by the Charge Point to the Central System. See also Firmware Status Notification.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	FirmwareStatus	1..1	Required. This contains the progress status of the firmware installation.

6.20. FirmwareStatusNotification.conf

This contains the field definition of the FirmwareStatusNotification.conf PDU sent by the Central System to the Charge Point in response to a FirmwareStatusNotification.req PDU. See also Firmware Status Notification.

No fields are defined.

6.21. GetCompositeSchedule.req

This contains the field definition of the GetCompositeSchedule.req PDU sent by the Central System to the Charge Point. See also Get Composite Schedule.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer	1..1	Required. The ID of the Connector for which the schedule is requested. When ConnectorId=0, the Charge Point will calculate the expected consumption for the grid connection.
duration	integer	1..1	Required. Time in seconds. length of requested schedule
chargingRateUnit	ChargingRateUnitType	0..1	Optional. Can be used to force a power or current profile

6.22. GetCompositeSchedule.conf

This contains the field definition of the GetCompositeSchedule.conf PDU sent by the Charge Point to the Central System in response to a GetCompositeSchedule.req PDU. See also Get Composite Schedule

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	GetCompositeScheduleStatus	1..1	Required. Status of the request. The Charge Point will indicate if it was able to process the request.
connectorId	integer	0..1	Optional. The charging schedule contained in this notification applies to a Connector.
scheduleStart	dateTime	0..1	Optional. Time. Periods contained in the charging profile are relative to this point in time. If status is "Rejected", this field may be absent.
chargingSchedule	ChargingSchedule	0..1	Optional. Planned Composite Charging Schedule, the energy consumption over time. Always relative to ScheduleStart. If status is "Rejected", this field may be absent.

6.23. GetConfiguration.req

This contains the field definition of the GetConfiguration.req PDU sent by the Central System to the Charge Point. See also Get Configuration.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
key	CiString50Type	0..*	Optional. List of keys for which the configuration value is requested.

6.24. GetConfiguration.conf

This contains the field definition of the GetConfiguration.conf PDU sent by Charge Point the to the Central System in response to a GetConfiguration.req. See also Get Configuration.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
configurationKey	KeyValue	0..*	Optional. List of requested or known keys
unknownKey	CiString50Type	0..*	Optional. Requested keys that are unknown

6.25. GetDiagnostics.req

This contains the field definition of the GetDiagnostics.req PDU sent by the Central System to the Charge Point. See also Get Diagnostics.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
location	anyURI	1..1	Required. This contains the location (directory) where the diagnostics file shall be uploaded to.
retries	integer	0..1	Optional. This specifies how many times Charge Point must try to upload the diagnostics before giving up. If this field is not present, it is left to Charge Point to decide how many times it wants to retry.
retryInterval	integer	0..1	Optional. The interval in seconds after which a retry may be attempted. If this field is not present, it is left to Charge Point to decide how long to wait between attempts.
startTime	dateTime	0..1	Optional. This contains the date and time of the oldest logging information to include in the diagnostics.
stopTime	dateTime	0..1	Optional. This contains the date and time of the latest logging information to include in the diagnostics.

6.26. GetDiagnostics.conf

This contains the field definition of the GetDiagnostics.conf PDU sent by the Charge Point to the Central System in response to a GetDiagnostics.req PDU. See also Get Diagnostics.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
fileName	CiString255Type	0..1	Optional. This contains the name of the file with diagnostic information that will be uploaded. This field is not present when no diagnostic information is available.

6.27. GetLocalListVersion.req

This contains the field definition of the GetLocalListVersion.req PDU sent by the Central System to the Charge Point. See also Get Local List Version.

No fields are defined.

6.28. GetLocalListVersion.conf

This contains the field definition of the GetLocalListVersion.conf PDU sent by the Charge Point to Central System in response to a GetLocalListVersion.req PDU. See also Get Local List Version.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
listVersion	integer	1..1	Required. This contains the current version number of the local authorization list in the Charge Point.

6.29. Heartbeat.req

This contains the field definition of the Heartbeat.req PDU sent by the Charge Point to the Central System. See also Heartbeat.

No fields are defined.

6.30. Heartbeat.conf

This contains the field definition of the Heartbeat.conf PDU sent by the Central System to the Charge Point in response to a Heartbeat.req PDU. See also Heartbeat.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
currentTime	dateTime	1..1	Required. This contains the current time of the Central System.

6.31. MeterValues.req

This contains the field definition of the MeterValues.req PDU sent by the Charge Point to the Central System. See also Meter Values.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer + connectorId >= 0	1..1	Required. This contains a number (>0) designating a connector of the Charge Point.'0' (zero) is used to designate the main powermeter.
transactionId	integer	0..1	Optional. The transaction to which these meter samples are related.
meterValue	MeterValue	1..*	Required. The sampled meter values with timestamps.

6.32. MeterValues.conf

This contains the field definition of the MeterValues.conf PDU sent by the Central System to the Charge Point in response to a MeterValues.req PDU. See also Meter Values.

No fields are defined.

6.33. RemoteStartTransaction.req

This contains the field definitions of the RemoteStartTransaction.req PDU sent to Charge Point by Central System. See also Remote Start Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer	0..1	Optional. Number of the connector on which to start the transaction. connectorId SHALL be > 0.
idTag	IdToken	1..1	Required. The identifier that Charge Point must use to start a transaction.
chargingProfile	ChargingProfile	0..1	Optional. Charging Profile to be used by the Charge Point for the requested transaction. ChargingProfilePurpose MUST be set to TxProfile.

6.34. RemoteStartTransaction.conf

This contains the field definitions of the RemoteStartTransaction.conf PDU sent from Charge Point to Central System. See also Remote Start Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	RemoteStartStopStatus	1..1	Required. Status indicating whether Charge Point accepts the request to start a transaction.

6.35. RemoteStopTransaction.req

This contains the field definitions of the RemoteStopTransaction.req PDU sent to Charge Point by Central System. See also Remote Stop Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
transactionId	integer	1..1	Required. The identifier of the transaction which Charge Point is requested to stop.

6.36. RemoteStopTransaction.conf

This contains the field definitions of the RemoteStopTransaction.conf PDU sent from Charge Point to Central System. See also Remote Stop Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	RemoteStartStopStatus	1..1	Required. Status indicating whether Charge Point accepts the request to stop a transaction.

6.37. ReserveNow.req

This contains the field definition of the ReserveNow.req PDU sent by the Central System to the Charge Point. See also Reserve Now.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer connectorId >= 0	1..1	Required. This contains the id of the connector to be reserved. A value of 0 means that the reservation is not for a specific connector.
expiryDate	dateTime	1..1	Required. This contains the date and time when the reservation ends.
idTag	IdToken	1..1	Required. The identifier for which the Charge Point has to reserve a connector.
parentIdTag	IdToken	0..1	Optional. The parent idTag.
reservationId	integer	1..1	Required. Unique id for this reservation.

6.38. ReserveNow.conf

This contains the field definition of the ReserveNow.conf PDU sent by the Charge Point to the Central System in response to a ReserveNow.req PDU. See also Reserve Now.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	ReservationStatus	1..1	Required. This indicates the success or failure of the reservation.

6.39. Reset.req

This contains the field definition of the Reset.req PDU sent by the Central System to the Charge Point. See also Reset.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
type	ResetType	1..1	Required. This contains the type of reset that the Charge Point should perform.

6.40. Reset.conf

This contains the field definition of the Reset.conf PDU sent by the Charge Point to the Central System in response to a Reset.req PDU. See also Reset.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	ResetStatus	1..1	Required. This indicates whether the Charge Point is able to perform the reset.

6.41. SendLocalList.req

This contains the field definition of the SendLocalList.req PDU sent by the Central System to the Charge Point.

If no (empty) localAuthorizationList is given and the updateType is Full, all identifications are removed from the list. Requesting a Differential update without (empty) localAuthorizationList will have no effect on the list. All idTags in the localAuthorizationList MUST be unique, no duplicate values are allowed. See also Send Local List.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
listVersion	integer	1..1	Required. In case of a full update this is the version number of the full list. In case of a differential update it is the version number of the list after the update has been applied.
localAuthorizationList	AuthorizationData	0..*	Optional. In case of a full update this contains the list of values that form the new local authorization list. In case of a differential update it contains the changes to be applied to the local authorization list in the Charge Point. Maximum number of AuthorizationData elements is available in the configuration key: `SendLocalListMaxLength`.
updateType	UpdateType	1..1	Required. This contains the type of update (full or differential) of this request.

6.42. SendLocalList.conf

This contains the field definition of the SendLocalList.conf PDU sent by the Charge Point to the Central System in response to a SendLocalList.req PDU. See also Send Local List.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	UpdateStatus	1..1	Required. This indicates whether the Charge Point has successfully received and applied the update of the local authorization list.

6.43. SetChargingProfile.req

This contains the field definition of the SetChargingProfile.req PDU sent by the Central System to the Charge Point.

The Central System uses this message to send charging profiles to a Charge Point. See also Set Charging Profile.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer	1..1	Required. The connector to which the charging profile applies. If connectorId = 0, the message contains an overall limit for the Charge Point.
csChargingProfiles	ChargingProfile	1..1	Required. The charging profile to be set at the Charge Point.

6.44. SetChargingProfile.conf

This contains the field definition of the SetChargingProfile.conf PDU sent by the Charge Point to the Central System in response to a SetChargingProfile.req PDU. See also Set Charging Profile.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	ChargingProfileStatus	1..1	Required. Returns whether the Charge Point has been able to process the message successfully. This does not guarantee the schedule will be followed to the letter. There might be other constraints the Charge Point may need to take into account.

6.45. StartTransaction.req

This section contains the field definition of the StartTransaction.req PDU sent by the Charge Point to the Central System. See also Start Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer connectorId > 0	1..1	Required. This identifies which connector of the Charge Point is used.
idTag	IdToken	1..1	Required. This contains the identifier for which a transaction has to be started.
meterStart	integer	1..1	Required. This contains the meter value in Wh for the connector at start of the transaction.
reservationId	integer	0..1	Optional. This contains the id of the reservation that terminates as a result of this transaction.
timestamp	dateTime	1..1	Required. This contains the date and time on which the transaction is started.

6.46. StartTransaction.conf

This contains the field definition of the StartTransaction.conf PDU sent by the Central System to the Charge Point in response to a StartTransaction.req PDU. See also Start Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
idTagInfo	IdTagInfo	1..1	Required. This contains information about authorization status, expiry and parent id.
transactionId	integer	1..1	Required. This contains the transaction id supplied by the Central System.

6.47. StatusNotification.req

This contains the field definition of the StatusNotification.req PDU sent by the Charge Point to the Central System. See also Status Notification.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer connectorId >= 0	1..1	Required. The id of the connector for which the status is reported. Id '0' (zero) is used if the status is for the Charge Point main controller.
errorCode	ChargePointErrorCode	1..1	Required. This contains the error code reported by the Charge Point.
info	CiString50Type	0..1	Optional. Additional free format information related to the error.
status	ChargePointStatus	1..1	Required. This contains the current status of the Charge Point.
timestamp	dateTime	0..1	Optional. The time for which the status is reported. If absent time of receipt of the message will be assumed.
vendorId	CiString255Type	0..1	Optional. This identifies the vendor-specific implementation.
vendorErrorCode	CiString50Type	0..1	Optional. This contains the vendor-specific error code.

6.48. StatusNotification.conf

This contains the field definition of the StatusNotification.conf PDU sent by the Central System to the Charge Point in response to an StatusNotification.req PDU. See also Status Notification.

No fields are defined.

6.49. StopTransaction.req

This contains the field definition of the StopTransaction.req PDU sent by the Charge Point to the Central System. See also Stop Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
idTag	IdToken	0..1	Optional. This contains the identifier which requested to stop the charging. It is optional because a Charge Point may terminate charging without the presence of an idTag, e.g. in case of a reset. A Charge Point SHALL send the idTag if known.
meterStop	integer	1..1	Required. This contains the meter value in Wh for the connector at end of the transaction.
timestamp	dateTime	1..1	Required. This contains the date and time on which the transaction is stopped.
transactionId	integer	1..1	Required. This contains the transaction-id as received by the StartTransaction.conf.
reason	Reason	0..1	Optional. This contains the reason why the transaction was stopped. MAY only be omitted when the Reason is "Local".

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
transactionData	MeterValue	0..*	Optional. This contains transaction usage details relevant for billing purposes.

6.50. StopTransaction.conf

This contains the field definition of the StopTransaction.conf PDU sent by the Central System to the Charge Point in response to a StopTransaction.req PDU. See also Stop Transaction.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
idTagInfo	IdTagInfo	0..1	Optional. This contains information about authorization status, expiry and parent id. It is optional, because a transaction may have been stopped without an identifier.

6.51. TriggerMessage.req

This contains the field definition of the TriggerMessage.req PDU sent by the Central System to the Charge Point. See also Trigger Message.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
requestedMessage	MessageTrigger	1..1	Required.
connectorId	integer connectorId > 0	0..1	Optional. Only filled in when request applies to a specific connector.

6.52. TriggerMessage.conf

This contains the field definition of the TriggerMessage.conf PDU sent by the Charge Point to the Central System in response to a TriggerMessage.req PDU. See also Trigger Message.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	TriggerMessageStatus	1..1	Required. Indicates whether the Charge Point will send the requested notification or not.

6.53. UnlockConnector.req

This contains the field definition of the UnlockConnector.req PDU sent by the Central System to the Charge Point. See also Unlock Connector.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
connectorId	integer connectorId > 0	1..1	Required. This contains the identifier of the connector to be unlocked.

6.54. UnlockConnector.conf

This contains the field definition of the UnlockConnector.conf PDU sent by the Charge Point to the Central System in response to an UnlockConnector.req PDU. See also Unlock Connector.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
status	UnlockStatus	1..1	Required. This indicates whether the Charge Point has unlocked the connector.

6.55. UpdateFirmware.req

This contains the field definition of the UpdateFirmware.req PDU sent by the Central System to the Charge Point. See also Update Firmware.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
location	anyURI	1..1	Required. This contains a string containing a URI pointing to a location from which to retrieve the firmware.
retries	integer	0..1	Optional. This specifies how many times Charge Point must try to download the firmware before giving up. If this field is not present, it is left to Charge Point to decide how many times it wants to retry.
retrieveDate	dateTime	1..1	Required. This contains the date and time after which the Charge Point is allowed to retrieve the (new) firmware.
retryInterval	integer	0..1	Optional. The interval in seconds after which a retry may be attempted. If this field is not present, it is left to Charge Point to decide how long to wait between attempts.

6.56. UpdateFirmware.conf

This contains the field definition of the UpdateFirmware.conf PDU sent by the Charge Point to the Central System in response to a UpdateFirmware.req PDU. See also Update Firmware.

No fields are defined.

Chapter 7. Types

7.1. AuthorizationData

Class

Elements that constitute an entry of a Local Authorization List update.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
idTag	IdToken	1..1	Required. The identifier to which this authorization applies.
idTagInfo	IdTagInfo	0..1	Optional. (Required when UpdateType is Full) This contains information about authorization status, expiry and parent id. For a Differential update the following applies: If this element is present, then this entry SHALL be added or updated in the Local Authorization List. If this element is absent, than the entry for this idtag in the Local Authorization List SHALL be deleted.

7.2. AuthorizationStatus

Enumeration

Status in a response to an Authorize.req.

VALUE	DESCRIPTION
Accepted	Identifier is allowed for charging.
Blocked	Identifier has been blocked. Not allowed for charging.
Expired	Identifier has expired. Not allowed for charging.
Invalid	Identifier is unknown. Not allowed for charging.
ConcurrentTx	Identifier is already involved in another transaction and multiple transactions are not allowed. (Only relevant for a StartTransaction.req.)

7.3. AvailabilityStatus

Enumeration

Status returned in response to ChangeAvailability.req.

VALUE	DESCRIPTION
Accepted	Request has been accepted and will be executed.
Rejected	Request has not been accepted and will not be executed.
Scheduled	Request has been accepted and will be executed when transaction(s) in progress have finished.

7.4. AvailabilityType

Enumeration

Requested availability change in ChangeAvailability.req.

VALUE	DESCRIPTION
Inoperative	Charge point is not available for charging.
Operative	Charge point is available for charging.

7.5. CancelReservationStatus

Enumeration

Status in CancelReservation.conf.

VALUE	DESCRIPTION
Accepted	Reservation for the identifier has been cancelled.
Rejected	Reservation could not be cancelled, because there is no reservation active for the identifier.

7.6. ChargePointErrorCode

Enumeration

Charge Point status reported in StatusNotification.req.

VALUE	DESCRIPTION
ConnectorLockFailure	Failure to lock or unlock connector.
EVCommunicationError	Communication failure with the vehicle, might be Mode 3 or other communication protocol problem. This is not a real error in the sense that the Charge Point doesn't need to go to the faulted state. Instead, it should go to the SuspendedEVSE state.
GroundFailure	Ground fault circuit interrupter has been activated.
HighTemperature	Temperature inside Charge Point is too high.
InternalError	Error in internal hard- or software component.
LocalListConflict	The authorization information received from the Central System is in conflict with the LocalAuthorizationList.
NoError	No error to report.
OtherError	Other type of error. More information in vendorErrorCode.
OverCurrentFailure	Over current protection device has tripped.
OverVoltage	Voltage has risen above an acceptable level.

VALUE	DESCRIPTION
PowerMeterFailure	Failure to read electrical/energy/power meter.
PowerSwitchFailure	Failure to control power switch.
ReaderFailure	Failure with idTag reader.
ResetFailure	Unable to perform a reset.
UnderVoltage	Voltage has dropped below an acceptable level.
WeakSignal	Wireless communication device reports a weak signal.

7.7. ChargePointStatus

Enumeration

Status reported in StatusNotification.req. A status can be reported for the Charge Point main controller (connectorId = 0) or for a specific connector. Status for the Charge Point main controller is a subset of the enumeration: *Available*, *Unavailable* or *Faulted*.

States considered Operative are: *Available*, *Preparing*, *Charging*, *SuspendedEVSE*, *SuspendedEV*, *Finishing*, *Reserved*.
States considered Inoperative are: *Unavailable*, *Faulted*.

STATUS	CONDITION
Available	When a Connector becomes available for a new user (Operative)
Preparing	When a Connector becomes no longer available for a new user but there is no ongoing Transaction (yet). Typically a Connector is in preparing state when a user presents a tag, inserts a cable or a vehicle occupies the parking bay (Operative)
Charging	When the contactor of a Connector closes, allowing the vehicle to charge (Operative)
SuspendedEVSE	When the EV is connected to the EVSE but the EVSE is not offering energy to the EV, e.g. due to a smart charging restriction, local supply power constraints, or as the result of StartTransaction.conf indicating that charging is not allowed etc. (Operative)
SuspendedEV	When the EV is connected to the EVSE and the EVSE is offering energy but the EV is not taking any energy. (Operative)
Finishing	When a Transaction has stopped at a Connector, but the Connector is not yet available for a new user, e.g. the cable has not been removed or the vehicle has not left the parking bay (Operative)
Reserved	When a Connector becomes reserved as a result of a Reserve Now command (Operative)
Unavailable	When a Connector becomes unavailable as the result of a Change Availability command or an event upon which the Charge Point transitions to unavailable at its discretion. Upon receipt of a Change Availability command, the status MAY change immediately or the change MAY be scheduled. When scheduled, the Status Notification shall be send when the availability change becomes effective (Inoperative)
Faulted	When a Charge Point or connector has reported an error and is not available for energy delivery. (Inoperative).

7.8. ChargingProfile

Class

A ChargingProfile consists of a ChargingSchedule, describing the amount of power or current that can be delivered per time interval.

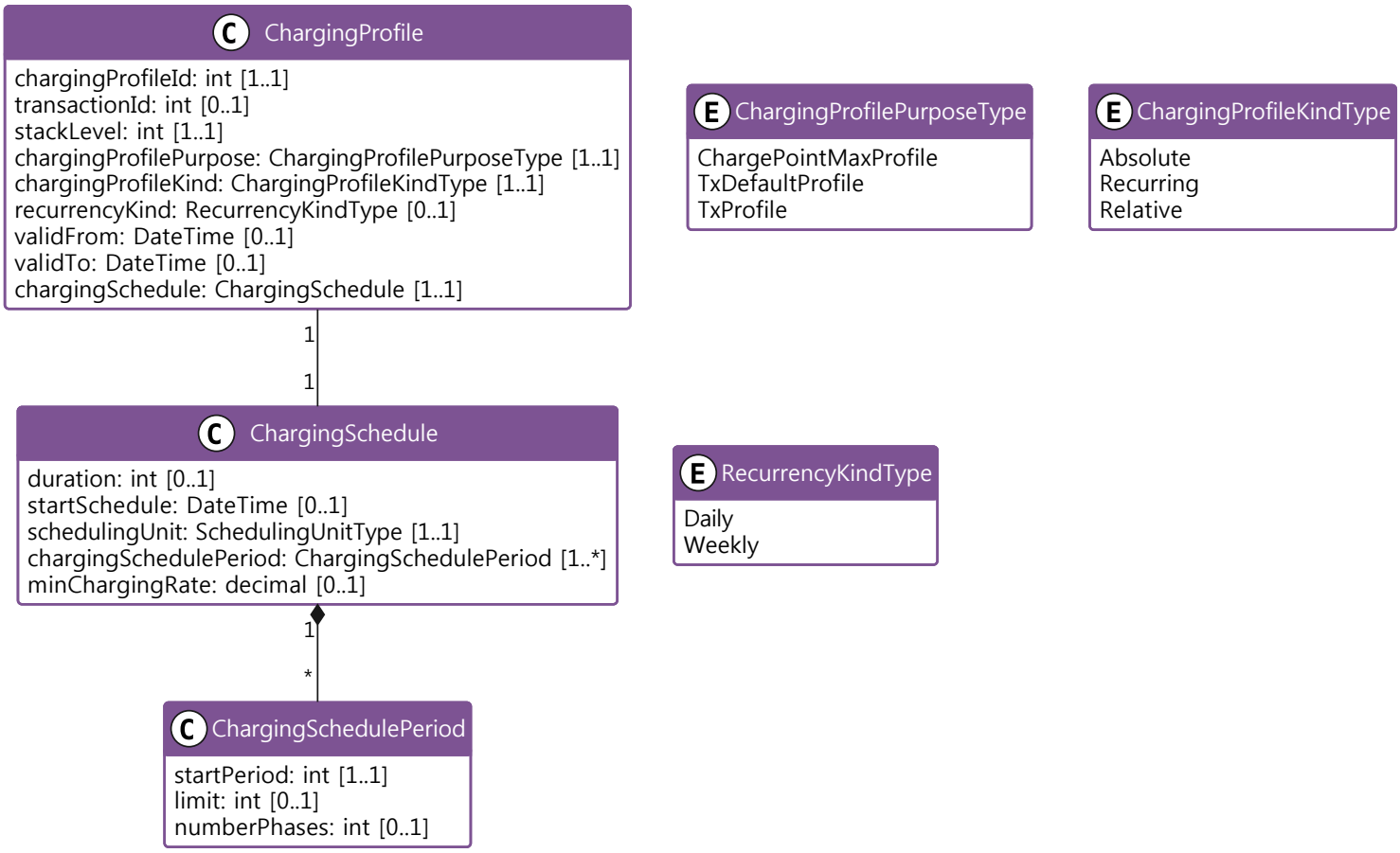

Figure 42. Class Diagram: ChargingProfile

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
chargingProfileId	integer	1..1	Required. Unique identifier for this profile.
transactionId	integer	0..1	Optional. Only valid if ChargingProfilePurpose is set to TxProfile, the transactionId MAY be used to match the profile to a specific transaction.
stackLevel	integer >=0	1..1	Required. Value determining level in hierarchy stack of profiles. Higher values have precedence over lower values. Lowest level is 0.
chargingProfilePurpose	ChargingProfilePurposeType	1..1	Required. Defines the purpose of the schedule transferred by this message.
chargingProfileKind	ChargingProfileKindType	1..1	Required. Indicates the kind of schedule.
recurrencyKind	RecurrencyKindType	0..1	Optional. Indicates the start point of a recurrence.
validFrom	dateTime	0..1	Optional. Point in time at which the profile starts to be valid. If absent, the profile is valid as soon as it is received by the Charge Point.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
validTo	dateTime	0..1	Optional. Point in time at which the profile stops to be valid. If absent, the profile is valid until it is replaced by another profile.
chargingSchedule	ChargingSchedule	1..1	Required. Contains limits for the available power or current over time.

7.9. ChargingProfileKindType

Enumeration

Kind of charging profile, as used in: ChargingProfile.

VALUE	DESCRIPTION
Absolute	Schedule periods are relative to a fixed point in time defined in the schedule.
Recurring	The schedule restarts periodically at the first schedule period.
Relative	Schedule periods are relative to a situation-specific start point (such as the start of a Transaction) that is determined by the charge point.

7.10. ChargingProfilePurposeType

Enumeration

Purpose of the charging profile, as used in: ChargingProfile.

VALUE	DESCRIPTION
ChargePointMaxProfile	Configuration for the maximum power or current available for an entire Charge Point.
TxDefaultProfile	Default profile *that can be configured in the Charge Point. When a new transaction is started, this profile SHALL be used, unless it was a transaction that was started by a RemoteStartTransaction.req with a ChargeProfile that is accepted by the Charge Point.
TxProfile	Profile with constraints to be imposed by the Charge Point on the current transaction, or on a new transaction when this is started via a RemoteStartTransaction.req with a ChargeProfile. A profile with this purpose SHALL cease to be valid when the transaction terminates.

7.11. ChargingProfileStatus

Enumeration

Status returned in response to SetChargingProfile.req.

VALUE	DESCRIPTION
Accepted	Request has been accepted and will be executed.
Rejected	Request has not been accepted and will not be executed.

VALUE	DESCRIPTION
NotSupported	Charge Point indicates that the request is not supported.

7.12. ChargingRateUnitType

Enumeration

Unit in which a charging schedule is defined, as used in: GetCompositeSchedule.req and ChargingSchedule.

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 not the measured voltage, but the set voltage for the area (hence, 230 of 110 volt). The "Number of Phases" is the numberPhases from the ChargingSchedulePeriod. It is usually more convenient to use this for DC charging. Note that if numberPhases in a ChargingSchedulePeriod 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.

7.13. ChargingSchedule

Class

Charging schedule structure defines a list of charging periods, as used in: GetCompositeSchedule.conf and ChargingProfile.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
duration	integer	0..1	Optional. Duration of the charging schedule in seconds. If the duration is left empty, the last period will continue indefinitely or until end of the transaction in case startSchedule is absent.
startSchedule	dateTime	0..1	Optional. Starting point of an absolute schedule. If absent the schedule will be relative to start of charging.
chargingRateUnit	ChargingRateUnitType	1..1	Required. The unit of measure Limit is expressed in.
chargingSchedulePeriod	ChargingSchedulePeriod	1..*	Required. List of ChargingSchedulePeriod elements defining maximum power or current usage over time. The startSchedule of the first ChargingSchedulePeriod SHALL always be 0.
minChargingRate	decimal	0..1	Optional. Minimum charging rate supported by the electric vehicle. 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).

7.14. ChargingSchedulePeriod

Class

Charging schedule period structure defines a time period in a charging schedule, as used in: ChargingSchedule.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
startPeriod	integer	1..1	Required. Start of the period, in seconds from the start of schedule. The value of StartPeriod also defines the stop time of the previous period.
limit	decimal	1..1	Required. Charging rate limit during the schedule period, in the applicable chargingRateUnit, for example in Amperes or Watts. Accepts at most one digit fraction (e.g. 8.1).
numberPhases	integer	0..1	Optional. The number of phases that can be used for charging. If a number of phases is needed, numberPhases=3 will be assumed unless another number is given.

7.15. CiString20Type

Type

Generic used case insensitive string of 20 characters.

FIELD TYPE	DESCRIPTION
CiString[20]	String is case insensitive.

7.16. CiString25Type

Type

Generic used case insensitive string of 25 characters.

FIELD TYPE	DESCRIPTION
CiString[25]	String is case insensitive.

7.17. CiString50Type

Type

Generic used case insensitive string of 50 characters.

FIELD TYPE	DESCRIPTION
CiString[50]	String is case insensitive.

7.18. CiString255Type

Type

Generic used case insensitive string of 255 characters.

FIELD TYPE	DESCRIPTION
CiString[255]	String is case insensitive.

7.19. CiString500Type

Type

Generic used case insensitive string of 500 characters.

FIELD TYPE	DESCRIPTION
CiString[500]	String is case insensitive.

7.20. ClearCacheStatus

Enumeration

Status returned in response to ClearCache.req.

VALUE	DESCRIPTION
Accepted	Command has been executed.
Rejected	Command has not been executed.

7.21. ClearChargingProfileStatus

Enumeration

Status returned in response to ClearChargingProfile.req.

VALUE	DESCRIPTION
Accepted	Request has been accepted and will be executed.
Unknown	No Charging Profile(s) were found matching the request.

7.22. ConfigurationStatus

Enumeration

Status in ChangeConfiguration.conf.

VALUE	DESCRIPTION
Accepted	Configuration key is supported and setting has been changed.
Rejected	Configuration key is supported, but setting could not be changed.

VALUE	DESCRIPTION
RebootRequired	Configuration key is supported and setting has been changed, but change will be available after reboot (Charge Point will not reboot itself).
NotSupported	Configuration key is not supported.

7.23. DataTransferStatus

Enumeration

Status in DataTransfer.conf.

VALUE	DESCRIPTION
Accepted	Message has been accepted and the contained request is accepted.
Rejected	Message has been accepted but the contained request is rejected.
UnknownMessageId	Message could not be interpreted due to unknown messageId string.
UnknownVendorId	Message could not be interpreted due to unknown vendorId string.

7.24. DiagnosticsStatus

Enumeration

Status in DiagnosticsStatusNotification.req.

VALUE	DESCRIPTION
Idle	Charge Point is not performing diagnostics related tasks. Status Idle SHALL only be used as in a DiagnosticsStatusNotification.req that was triggered by a TriggerMessage.req.
Uploaded	Diagnostics information has been uploaded.
UploadFailed	Uploading of diagnostics failed.
Uploading	File is being uploaded.

7.25. FirmwareStatus

Enumeration

Status of a firmware download as reported in FirmwareStatusNotification.req.

VALUE	DESCRIPTION
Downloaded	New firmware has been downloaded by Charge Point.
DownloadFailed	Charge point failed to download firmware.

VALUE	DESCRIPTION
Downloading	Firmware is being downloaded.
Idle	Charge Point is not performing firmware update related tasks. Status Idle SHALL only be used as in a FirmwareStatusNotification.req that was triggered by a TriggerMessage.req.
InstallationFailed	Installation of new firmware has failed.
Installing	Firmware is being installed.
Installed	New firmware has successfully been installed in charge point.

7.26. GetCompositeScheduleStatus

Enumeration

Status returned in response to GetCompositeSchedule.req.

VALUE	DESCRIPTION
Accepted	Request has been accepted and will be executed.
Rejected	Request has not been accepted and will not be executed.

7.27. IdTagInfo

Class

Contains status information about an identifier. It is returned in Authorize, Start Transaction and Stop Transaction responses.

If expiryDate is not given, the status has no end date.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
expiryDate	dateTime	0..1	Optional. This contains the date at which idTag should be removed from the Authorization Cache.
parentIdTag	IdToken	0..1	Optional. This contains the parent-identifier.
status	AuthorizationStatus	1..1	Required. This contains whether the idTag has been accepted or not by the Central System.

7.28. IdToken

Type

Contains the identifier to use for authorization. It is a case insensitive string. In future releases this may become a complex type to support multiple forms of identifiers.

FIELD TYPE	DESCRIPTION
CiString20Type	IdToken is case insensitive.

7.29. KeyValue

Class

Contains information about a specific configuration key. It is returned in GetConfiguration.conf.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
key	CiString50Type	1..1	Required.
readonly	boolean	1..1	Required. False if the value can be set with the ChangeConfiguration message
value	CiString500Type	0..1	Optional. If key is known but not set, this field may be absent

7.30. Location

Enumeration

Allowable values of the optional "location" field of a value element in SampledValue.

VALUE	DESCRIPTION
Body	Measurement inside body of Charge Point (e.g. Temperature)
Cable	Measurement taken from cable between EV and Charge Point
EV	Measurement taken by EV
Inlet	Measurement at network ("grid") inlet connection
Outlet	Measurement at a Connector. Default value

7.31. Measurand

Enumeration

Allowable values of the optional "measurand" field of a Value element, as used in MeterValues.req and StopTransaction.req messages. Default value of "measurand" is always "Energy.Active.Import.Register".

Import is energy flow from the Grid to the Charge Point, EV or other load. Export is energy flow from the EV to the Charge Point and/or from the Charge Point to the Grid.

VALUE	DESCRIPTION
Current.Export	Instantaneous current flow from EV

VALUE	DESCRIPTION
Current.Import	Instantaneous current flow to EV
Current.Offered	Maximum current offered to EV
Energy.Active.Export.Register	Numerical value read from the "active electrical energy" (Wh or kWh) register of the (most authoritative) electrical meter measuring energy exported (to the grid).
Energy.Active.Import.Register	Numerical value read from the "active electrical energy" (Wh or kWh) register of the (most authoritative) electrical meter measuring energy imported (from the grid supply).
Energy.Reactive.Export.Register	Numerical value read from the "reactive electrical energy" (VARh or kVARh) register of the (most authoritative) electrical meter measuring energy exported (to the grid).
Energy.Reactive.Import.Register	Numerical value read from the "reactive electrical energy" (VARh or kVARh) register of the (most authoritative) electrical meter measuring energy imported (from the grid supply).
Energy.Active.Export.Interval	Absolute amount of "active electrical energy" (Wh or kWh) exported (to the grid) during an associated time "interval", specified by a Metervalues ReadingContext, and applicable interval duration configuration values (in seconds) for "ClockAlignedDataInterval" and "MeterValueSampleInterval".
Energy.Active.Import.Interval	Absolute amount of "active electrical energy" (Wh or kWh) imported (from the grid supply) during an associated time "interval", specified by a Metervalues ReadingContext, and applicable interval duration configuration values (in seconds) for "ClockAlignedDataInterval" and "MeterValueSampleInterval".
Energy.Reactive.Export.Interval	Absolute amount of "reactive electrical energy" (VARh or kVARh) exported (to the grid) during an associated time "interval", specified by a Metervalues ReadingContext, and applicable interval duration configuration values (in seconds) for "ClockAlignedDataInterval" and "MeterValueSampleInterval".
Energy.Reactive.Import.Interval	Absolute amount of "reactive electrical energy" (VARh or kVARh) imported (from the grid supply) during an associated time "interval", specified by a Metervalues ReadingContext, and applicable interval duration configuration values (in seconds) for "ClockAlignedDataInterval" and "MeterValueSampleInterval".
Frequency	Instantaneous reading of powerline frequency. NOTE: OCPP 1.6 does not have a UnitOfMeasure for frequency, the UnitOfMeasure for any SampledValue with measurand: Frequency is Hertz.
Power.Active.Export	Instantaneous active power exported by EV. (W or kW)
Power.Active.Import	Instantaneous active power imported by EV. (W or kW)
Power.Factor	Instantaneous power factor of total energy flow
Power.Offered	Maximum power offered to EV
Power.Reactive.Export	Instantaneous reactive power exported by EV. (var or kvar)
Power.Reactive.Import	Instantaneous reactive power imported by EV. (var or kvar)
RPM	Fan speed in RPM
SoC	State of charge of charging vehicle in percentage
Temperature	Temperature reading inside Charge Point.
Voltage	Instantaneous AC RMS supply voltage

All "Register" values relating to a single charging transaction, or a non-transactional consumer (e.g. charge point internal power supply, overall supply) MUST be monotonically increasing in time.

The actual quantity of energy corresponding to a reported "Register" value is computed as the register value in question minus the register value recorded/reported at the start of the transaction or other relevant starting reference point in time. For improved auditability "Register" values SHOULD reported exactly as they are directly read from a non-volatile register in the electrical metering hardware, and SHOULD NOT be re-based to zero at the start of transactions. This allows any "missing energy" between sequential transactions, due to hardware fault, mis-wiring, fraud, etc. to be identified, by allowing the Central System to confirm that the starting register value of any transaction is identical to the finishing register value of the preceding transaction on the same connector.

7.32. MessageTrigger

Enumeration

Type of request to be triggered in a TriggerMessage.req.

VALUE	DESCRIPTION
BootNotification	To trigger a BootNotification request
DiagnosticsStatusNotification	To trigger a DiagnosticsStatusNotification request
FirmwareStatusNotification	To trigger a FirmwareStatusNotification request
Heartbeat	To trigger a Heartbeat request
MeterValues	To trigger a MeterValues request
StatusNotification	To trigger a StatusNotification request

7.33. MeterValue

Class

Collection of one or more sampled values in MeterValues.req and StopTransaction.req. All sampled values in a MeterValue are sampled at the same point in time.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
timestamp	dateTime	1..1	Required. Timestamp for measured value(s)
sampledValue	SampledValue	1..*	Required. One or more measured values

7.34. Phase

Enumeration

Phase as used in SampledValue. Phase specifies how a measured value is to be interpreted. Please note that not all values of Phase are applicable to all Measurands.

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

7.35. ReadingContext

Enumeration

Values of the context field of a value in SampledValue.

VALUE	DESCRIPTION
Interruption.Begin	Value taken at start of interruption.
Interruption.End	Value taken when resuming after interruption.
Other	Value for any other situations.
Sample.Clock	Value taken at clock aligned interval.
Sample.Periodic	Value taken as periodic sample relative to start time of transaction.
Transaction.Begin	Value taken at start of transaction.
Transaction.End	Value taken at end of transaction.
Trigger	Value taken in response to a TriggerMessage.req.

7.36. Reason

Enumeration

Reason for stopping a transaction in StopTransaction.req.

VALUE	DESCRIPTION
DeAuthorized	The transaction was stopped because of the authorization status in a StartTransaction.conf.
EmergencyStop	Emergency stop button was used.
EVDisconnected	disconnecting of cable, vehicle moved away from inductive charge unit.
HardReset	A hard reset command was received.
Local	Stopped locally on request of the user at the Charge Point. This is a regular termination of a transaction. Examples: presenting an RFID tag, pressing a button to stop.
Other	Any other reason.
PowerLoss	Complete loss of power.
Reboot	A locally initiated reset/reboot occurred. (for instance watchdog kicked in)
Remote	Stopped remotely on request of the user. This is a regular termination of a transaction. Examples: termination using a smartphone app, exceeding a (non local) prepaid credit.
SoftReset	A soft reset command was received.
UnlockCommand	Central System sent an Unlock Connector command.

7.37. RecurrencyKindType

Enumeration

Type of recurrence of a charging profile, as used in ChargingProfile.

VALUE	DESCRIPTION
Daily	The schedule restarts every 24 hours, at the same time as in the startSchedule.
Weekly	The schedule restarts every 7 days, at the same time and day-of-the-week as in the startSchedule.

7.38. RegistrationStatus

Enumeration

Result of registration in response to BootNotification.req.

VALUE	DESCRIPTION
Accepted	Charge point is accepted by Central System.
Pending	Central System is not yet ready to accept the Charge Point. Central System may send messages to retrieve information or prepare the Charge Point.
Rejected	Charge point is not accepted by Central System. This may happen when the Charge Point id is not known by Central System.

7.39. RemoteStartStopStatus

Enumeration

The result of a RemoteStartTransaction.req or RemoteStopTransaction.req request.

VALUE	DESCRIPTION
Accepted	Command will be executed.
Rejected	Command will not be executed.

7.40. ReservationStatus

Enumeration

Status in ReserveNow.conf.

VALUE	DESCRIPTION
Accepted	Reservation has been made.
Faulted	Reservation has not been made, because connectors or specified connector are in a faulted state.
Occupied	Reservation has not been made. All connectors or the specified connector are occupied.
Rejected	Reservation has not been made. Charge Point is not configured to accept reservations.
Unavailable	Reservation has not been made, because connectors or specified connector are in an unavailable state.

7.41. ResetStatus

Enumeration

Result of Reset.req.

VALUE	DESCRIPTION
Accepted	Command will be executed.
Rejected	Command will not be executed.

7.42. ResetType

Enumeration

Type of reset requested by Reset.req.

VALUE	DESCRIPTION
Hard	Restart (all) the hardware, the Charge Point is not required to gracefully stop ongoing transaction. If possible the Charge Point sends a StopTransaction.req for previously ongoing transactions after having restarted and having been accepted by the Central System via a BootNotification.conf. This is a last resort solution for a not correctly functioning Charge Point, by sending a "hard" reset, (queued) information might get lost.
Soft	Stop ongoing transactions gracefully and sending StopTransaction.req for every ongoing transaction. It should then restart the application software (if possible, otherwise restart the processor/controller).

7.43. SampledValue

Class

Single sampled value in MeterValues. Each value can be accompanied by optional fields.

FIELD NAME	FIELD TYPE	CARD.	DESCRIPTION
value	String	1..1	Required. Value as a "Raw" (decimal) number or "SignedData". Field Type is "string" to allow for digitally signed data readings. Decimal numeric values are also acceptable to allow fractional values for measurands such as Temperature and Current.
context	ReadingContext	0..1	Optional. Type of detail value: start, end or sample. Default = "Sample.Periodic"
format	ValueFormat	0..1	Optional. Raw or signed data. Default = "Raw"
measurand	Measurand	0..1	Optional. Type of measurement. Default = "Energy.Active.Import.Register"
phase	Phase	0..1	Optional. indicates how the measured value is to be interpreted. For instance between L1 and neutral (L1-N) Please note that not all values of phase are applicable to all Measurands. When phase is absent, the measured value is interpreted as an overall value.
location	Location	0..1	Optional. Location of measurement. Default="Outlet"
unit	UnitOfMeasure	0..1	Optional. Unit of the value. Default = "Wh" if the (default) measurand is an "Energy" type.

7.44. TriggerMessageStatus

Enumeration

Status in TriggerMessage.conf.

VALUE	DESCRIPTION
Accepted	Requested notification will be sent.
Rejected	Requested notification will not be sent.
NotImplemented	Requested notification cannot be sent because it is either not implemented or unknown.

7.45. UnitOfMeasure

Enumeration

Allowable values of the optional "unit" field of a Value element, as used in SampledValue. Default value of "unit" is always "Wh".

VALUE	DESCRIPTION
Wh	Watt-hours (energy). Default.
kWh	kiloWatt-hours (energy).
varh	Var-hours (reactive energy).
kvarh	kilovar-hours (reactive energy).
W	Watts (power).
kW	kilowatts (power).
VA	VoltAmpere (apparent power).
kVA	kiloVolt Ampere (apparent power).
var	Vars (reactive power).
kvar	kilovars (reactive power).
A	Amperes (current).
V	Voltage (r.m.s. AC).
Celsius	Degrees (temperature).
Fahrenheit	Degrees (temperature).
K	Degrees Kelvin (temperature).
Percent	Percentage.

7.46. UnlockStatus

Enumeration

Status in response to UnlockConnector.req.

VALUE	DESCRIPTION
Unlocked	Connector has successfully been unlocked.
UnlockFailed	Failed to unlock the connector: The Charge Point has tried to unlock the connector and has detected that the connector is still locked or the unlock mechanism failed.
NotSupported	Charge Point has no connector lock, or ConnectorId is unknown.

7.47. UpdateStatus

Enumeration

Type of update for a SendLocalList.req.

VALUE	DESCRIPTION
Accepted	Local Authorization List successfully updated.
Failed	Failed to update the Local Authorization List.
NotSupported	Update of Local Authorization List is not supported by Charge Point.
VersionMismatch	Version number in the request for a differential update is less or equal then version number of current list.

7.48. UpdateType

Enumeration

Type of update for a SendLocalList.req.

VALUE	DESCRIPTION
Differential	Indicates that the current Local Authorization List must be updated with the values in this message.
Full	Indicates that the current Local Authorization List must be replaced by the values in this message.

7.49. ValueFormat

Enumeration

Format that specifies how the value element in SampledValue is to be interpreted.

VALUE	DESCRIPTION
Raw	Data is to be interpreted as integer/decimal numeric data.
SignedData	Data is represented as a signed binary data block, encoded as hex data.

Chapter 8. Firmware and Diagnostics File Transfer

This section is normative.

The supported transfer protocols are controlled by the configuration key *SupportedFileTransferProtocols*. FTP, FTPS, HTTP, HTTPS (CSL)

8.1. Download Firmware

When a Charge Point is notified about new firmware, it needs to be able to download this firmware. The Central System supplies in the request an URL where the firmware can be downloaded. The URL also contains the protocol which must be used to download the firmware.

It is recommended that the firmware is downloaded via FTP or FTPS. FTP(S) is better optimized for large binary data than HTTP. Also FTP(S) has the ability to resume downloads. In case a download is interrupted, the Charge Point can resume downloading after the part it already has downloaded. The FTP URL is of format: *ftp://user:password@host:port/path* in which the parts *user:password@*, *:password* or *:port* may be excluded.

To ensure that the correct firmware is downloaded, it is RECOMMENDED that the firmware is also digitally signed.

8.2. Upload Diagnostics

When a Charge Point is requested to upload a diagnostics file, the Central System supplies in the request an URL where the Charge Point should upload the file. The URL also contains the protocol which must be used to upload the file.

It is recommended that the diagnostics file is downloaded via FTP or FTPS. FTP(S) is better optimized for large binary data than HTTP. Also FTP(S) has the ability to resume uploads. In case an upload is interrupted, the Charge Point can resume uploading after the part it already has uploaded. The FTP URL is of format: *ftp://user:password@host:port/path* in which the parts *user:password@*, *:password* or *:port* may be excluded.

Chapter 9. Standard Configuration Key Names & Values

Below follows a list of all configuration keys with a role standardized in this specification. The list is separated by Feature Profiles. A required configuration key mentioned under a particular profile only has to be supported by the Charge Point if it supports that profile.

For optional Configuration Keys with a boolean type, the following rules apply for the configuration key in the response to a GetConfiguration.req without a list of keys:

- If the key is present, the Charge Point provides the functionality that is configured by the key, and it can be enabled or disabled by setting the value for the key.
- If the key is not present, the Charge Point does not provide the functionality that can be configured by the key.

The "Accessibility" property shows if the value for a certain configuration key is read-only ("R") or read-write ("RW"). In case the key is read-only, the Central System can read the value for the key using GetConfiguration, but not write it. In case the accessibility is read-write, the Central System can also write the value for the key using ChangeConfiguration.

9.1. Core Profile

9.1.1. AllowOfflineTxForUnknownId

Required/optional	optional
Accessibility	RW
Type	boolean
Description	If this key exists, the Charge Point supports Unknown Offline Authorization. If this key reports a value of *true*, Unknown Offline Authorization is enabled.

9.1.2. AuthorizationCacheEnabled

Required/optional	optional
Accessibility	RW
Type	boolean
Description	If this key exists, the Charge Point supports an Authorization Cache. If this key reports a value of *true* , the Authorization Cache is enabled.

9.1.3. AuthorizeRemoteTxRequests

Required/optional	required
Accessibility	R or RW. Choice is up to Charge Point implementation.
Type	boolean

Description	Whether a remote request to start a transaction in the form of a RemoteStartTransaction.req message should be authorized beforehand like a local action to start a transaction.

9.1.4. BlinkRepeat

Required/optional	optional
Accessibility	RW
Type	integer
Unit	times
Description	Number of times to blink Charge Point lighting when signalling

9.1.5. ClockAlignedDataInterval

Required/optional	required
Accessibility	RW
Type	integer
Unit	seconds
Description	Size (in seconds) of the clock-aligned data interval. This is the size (in seconds) of the set of evenly spaced aggregation intervals per day, starting at 00:00:00 (midnight). For example, a value of 900 (15 minutes) indicates that every day should be broken into 96 15-minute intervals. When clock aligned data is being transmitted, the interval in question is identified by the start time and (optional) duration interval value, represented according to the ISO8601 standard. All "per-period" data (e.g. energy readings) should be accumulated (for "flow" type measurands such as energy), or averaged (for other values) across the entire interval (or partial interval, at the beginning or end of a Transaction), and transmitted (if so enabled) at the end of each interval, bearing the interval start time timestamp. A value of "0" (numeric zero), by convention, is to be interpreted to mean that no clock-aligned data should be transmitted.

9.1.6. ConnectionTimeOut

Required/optional	required
Accessibility	RW
Type	integer
Unit	seconds
Description	Interval *from beginning of status: 'Preparing' until incipient Transaction is automatically canceled, due to failure of EV driver to (correctly) insert the charging cable connector(s) into the appropriate socket(s). The Charge Point SHALL go back to the original state, probably: 'Available'.

9.1.7. ConnectorPhaseRotation

Required/optional	required
Accessibility	RW
Type	CSL
Description	The phase rotation per connector in respect to the connector's electrical meter (or if absent, the grid connection). Possible values per connector are: `NotApplicable` (for Single phase or DC Charge Points) `Unknown` (not (yet) known) `RST` (Standard Reference Phasing) `RTS` (Reversed Reference Phasing) `SRT` (Reversed 240 degree rotation) `STR` (Standard 120 degree rotation) `TRS` (Standard 240 degree rotation) `TSR` (Reversed 120 degree rotation) R can be identified as phase 1 (L1), S as phase 2 (L2), T as phase 3 (L3). If known, the Charge Point MAY also report the phase rotation between the grid connection and the main energymeter by using index number Zero (0). Values are reported in CSL, formatted: 0.RST, 1.RST, 2.RTS

9.1.8. ConnectorPhaseRotationMaxLength

Required/optional	optional
Accessibility	R
Type	integer
Description	Maximum number of items in a `ConnectorPhaseRotation` Configuration Key.

9.1.9. GetConfigurationMaxKeys

Required/optional	required
Accessibility	R
Type	integer
Description	Maximum number of requested configuration keys in a GetConfiguration.req PDU.

9.1.10. HeartbeatInterval

Required/optional	required

Accessibility	RW
Type	integer
Unit	seconds
Description	Interval of inactivity (no OCPP exchanges) with central system after which the Charge Point should send a Heartbeat.req PDU

9.1.11. LightIntensity

Required/optional	optional
Accessibility	RW
Type	integer
Unit	%
Description	Percentage of maximum intensity at which to illuminate Charge Point lighting

9.1.12. LocalAuthorizeOffline

Required/optional	required
Accessibility	RW
Type	boolean
Description	whether the Charge Point, when *offline*, will start a transaction for locally-authorized identifiers.

9.1.13. LocalPreAuthorize

Required/optional	required
Accessibility	RW
Type	boolean
Description	whether the Charge Point, when online, will start a transaction for locally-authorized identifiers without waiting for or requesting an Authorize.conf from the Central System

9.1.14. MaxEnergyOnInvalidId

Required/optional	optional
Accessibility	RW
Type	integer

Unit	Wh
Description	Maximum energy in Wh delivered when an identifier is invalidated by the Central System after start of a transaction.

9.1.15. MeterValuesAlignedData

Required/optional	required
Accessibility	RW
Type	CSL
Description	Clock-aligned measurand(s) to be included in a MeterValues.req PDU, every ClockAlignedDataInterval seconds

9.1.16. MeterValuesAlignedDataMaxLength

Required/optional	optional
Accessibility	R
Type	integer
Description	Maximum number of items in a MeterValuesAlignedData Configuration Key.

9.1.17. MeterValuesSampledData

Required/optional	required
Accessibility	RW
Type	CSL
Description	Sampled measurands to be included in a MeterValues.req PDU, every MeterValueSampleInterval seconds. Where applicable, the Measurand is combined with the optional phase; for instance: Voltage.L1 Default: "Energy.Active.Import.Register"

9.1.18. MeterValuesSampledDataMaxLength

Required/optional	optional
Accessibility	R
Type	integer
Description	Maximum number of items in a MeterValuesSampledData Configuration Key.

9.1.19. MeterValueSampleInterval

Required/optional	required
Accessibility	RW
Type	integer
Unit	seconds
Description	Interval between sampling of metering (or other) data, intended to be transmitted by "MeterValues" PDUs. For charging session data (ConnectorId>0), samples are acquired and transmitted periodically at this interval from the start of the charging transaction. A value of "0" (numeric zero), by convention, is to be interpreted to mean that no sampled data should be transmitted.

9.1.20. MinimumStatusDuration

Required/optional	optional
Accessibility	RW
Type	integer
Unit	seconds
Description	The minimum duration that a Charge Point or Connector status is stable before a StatusNotification.req PDU is sent to the Central System.

9.1.21. NumberOfConnectors

Required/optional	required
Accessibility	R
Type	integer
Description	The number of physical charging connectors of this Charge Point.

9.1.22. ResetRetries

Required/optional	required
Accessibility	RW
Type	integer
Unit	times
Description	Number of times to retry an unsuccessful reset of the Charge Point.

9.1.23. StopTransactionOnEVSideDisconnect

Required/optional	required
Accessibility	RW
Type	boolean
Description	When set to *true*, the Charge Point SHALL administratively stop the transaction when the cable is unplugged from the EV.

9.1.24. StopTransactionOnInvalidId

Required/optional	required
Accessibility	RW
Type	boolean
Description	whether the Charge Point will stop an ongoing transaction when it receives a non- *Accepted* authorization status in a StartTransaction.conf for this transaction

9.1.25. StopTxnAlignedData

Required/optional	required
Accessibility	RW
Type	CSL
Description	Clock-aligned periodic measurand(s) to be included in the TransactionData element of StopTransaction.req MeterValues.req PDU for every `ClockAlignedDataInterval` of the Transaction

9.1.26. StopTxnAlignedDataMaxLength

Required/optional	optional
Accessibility	R
Type	integer
Description	Maximum number of items in a `StopTxnAlignedData` Configuration Key.

9.1.27. StopTxnSampledData

Required/optional	required
Accessibility	RW

Type	CSL
Description	Sampled measurands to be included in the TransactionData element of StopTransaction.req PDU, every `MeterValueSampleInterval` seconds from the start of the charging session

9.1.28. StopTxnSampledDataMaxLength

Required/optional	optional
Accessibility	R
Type	integer
Description	Maximum number of items in a `StopTxnSampledData` Configuration Key.

9.1.29. SupportedFeatureProfiles

Required/optional	required
Accessibility	R
Type	CSL
Description	A list of supported Feature Profiles. Possible profile identifiers: Core, FirmwareManagement, LocalAuthListManagement, Reservation, SmartCharging and RemoteTrigger.

9.1.30. SupportedFeatureProfilesMaxLength

Required/optional	optional
Accessibility	R
Type	integer
Description	Maximum number of items in a `SupportedFeatureProfiles` Configuration Key.

9.1.31. TransactionMessageAttempts

Required/optional	required
Accessibility	RW
Type	integer
Unit	times
Description	How often the Charge Point should try to submit a transaction-related message when the Central System fails to process it.

9.1.32. TransactionMessageRetryInterval

Required/optional	required
Accessibility	RW
Type	integer
Unit	seconds
Description	How long the Charge Point should wait before resubmitting a transaction-related message that the Central System failed to process.

9.1.33. UnlockConnectorOnEVSideDisconnect

Required/optional	required
Accessibility	RW
Type	boolean
Description	When set to *true* , the Charge Point SHALL unlock the cable on Charge Point side when the cable is unplugged at the EV.

9.1.34. WebSocketPingInterval

Required/optional	optional
Accessibility	RW
Type	integer
Unit	seconds
Description	Only relevant for websocket implementations. 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. ChangeConfiguration is expected to return a REJECTED result.

9.2. Local Auth List Management Profile

9.2.1. LocalAuthListEnabled

Required/optional	required
Accessibility	RW
Type	boolean
Description	whether the Local Authorization List is enabled

9.2.2. LocalAuthListMaxLength

Required/optional	required
Accessibility	R
Type	integer
Description	Maximum number of identifications that can be stored in the Local Authorization List

9.2.3. SendLocalListMaxLength

Required/optional	required
Accessibility	R
Type	integer
Description	Maximum number of identifications that can be send in a single SendLocalList.req

9.3. Reservation Profile

9.3.1. ReserveConnectorZeroSupported

Required/optional	optional
Accessibility	R
Type	boolean
Description	If this configuration key is present and set to *true*: Charge Point support reservations on connector 0.

9.4. Smart Charging Profile

9.4.1. ChargeProfileMaxStackLevel

Required/optional	required
Accessibility	R
Type	integer
Description	Max StackLevel of a ChargingProfile. The number defined also indicates the max allowed number of installed charging schedules per Charging Profile Purposes.

9.4.2. ChargingScheduleAllowedChargingRateUnit

Required/optional	required

Accessibility	R
Type	CSL
Description	A list of supported quantities for use in a ChargingSchedule. Allowed values: 'Current' and 'Power'

9.4.3. ChargingScheduleMaxPeriods

Required/optional	required
Accessibility	R
Type	integer
Description	Maximum number of periods that may be defined per ChargingSchedule.

9.4.4. ConnectorSwitch3to1PhaseSupported

Required/optional	optional
Accessibility	R
Type	boolean
Description	If defined and true, this Charge Point support switching from 3 to 1 phase during a Transaction.

9.4.5. MaxChargingProfilesInstalled

Required/optional	required
Accessibility	R
Type	integer
Description	Maximum number of Charging profiles installed at a time

Appendix A: New in OCPP 1.6

The following changes are made in OCPP 1.6 compared to OCPP 1.5 [OCPP1.5]:

- Smart Charging is added
- A binding to JSON over WebSocket as a transport protocol is added, reducing data usage and enabling OCPP communication through NAT routers, see: OCPP JSON Specification
- Extra statuses are added to the ChargePointStatus enumeration, giving the CPO and ultimately end-users more information about the current status of a Charge Point
- Structure of MeterValues.req is changed to eliminate use of XML Attributes, this is needed for support of JSON (no attribute support in JSON).
- Extra values are added to the Measurand enumeration, giving Charge Point manufacturers the possibility to send new information to a Central System, such as the State of Charge of an EV
- The TriggerMessage message is added, giving the Central System the possibility to request information from the Charge Point
- A new *Pending* member is added to the RegistrationStatus enumeration used in BootNotification.conf
- More and clearer configuration keys are added, making it clearer to the CPO how to configure the different business cases in a Charge Point
- The messages and configuration keys are split into profiles, making it easier to implement OCPP gradually or only in part
- Known ambiguities are removed (e.g. when to use UnlockConnector.req, how to respond to RemoteStart/Stop, Connector numbering)

A.1. Updated/New Messages

- BootNotification.req
  - Change IccId and Imsi to CiString[] to enforce maximum lengths.
- BootNotification.conf
  - heartbeatInterval to interval, interval now also used for other purposes than heartbeat, need to fix in spec
  - Added status Pending
- ChargePointErrorCode
  - Added enumvalues: `InternalError`, `LocalListConfict` and `UnderVoltage`
  - Renamed enum value `Mode3Error` to `EVCommunicationError`
- ChargePointStatus
  - Replaced enum value `Occupied` with the more detailed values: `Preparing`, `Charging`, `SuspendedEVSE`, `SuspendedEV` and `Finishing`
- ChargingRateUnitType
  - New
- ConfigurationStatus
  - Added enum RebootRequired
- ClearChargingProfile.req
  - New
- ClearChargingProfile.conf
  - New
- DiagnosticsStatus

- Added enum Uploading and Idle

- FirmwareStatus
  - Added enum Downloading, Installing and Idle
- GetCompositeSchedule.req
  - New
- GetCompositeSchedule.conf
  - New
- Location
  - Added enum Cable and EV
- Measurand
  - Added enum `Current.Offered`, `Frequency`, `Power.Factor`, `Power.Offered`, `RPM` and `SoC`
- MeterValues.req
  - overhaul of complex data structures
  - Added 'phase' field
- ReadingContext
  - Added enum `Trigger` and `Other`
- RemoteStartTransaction.req
  - Added ChargingProfile optional
- SendLocalList.req
  - removed hash
- SendLocalList.conf
  - removed hash
- SetChargingProfile.req
  - New
- SetChargingProfile.conf
  - New
- StatusNotification.req
  - Overhaul of states
  - New error codes
  - Connector id 0 can only have status: Available, Unavailable and Faulted.
- StopTransaction.req
  - added explicit and required stop reason
- TriggerMessage.req
  - New
- TriggerMessage.conf
  - New
- UnlockConnector.conf
  - overhaul of UnlockStatus enum
- UnitOfMeasure

- Added Fahrenheit, K, Percent, VA, kVA
- Rename Volt to V, Amp to A