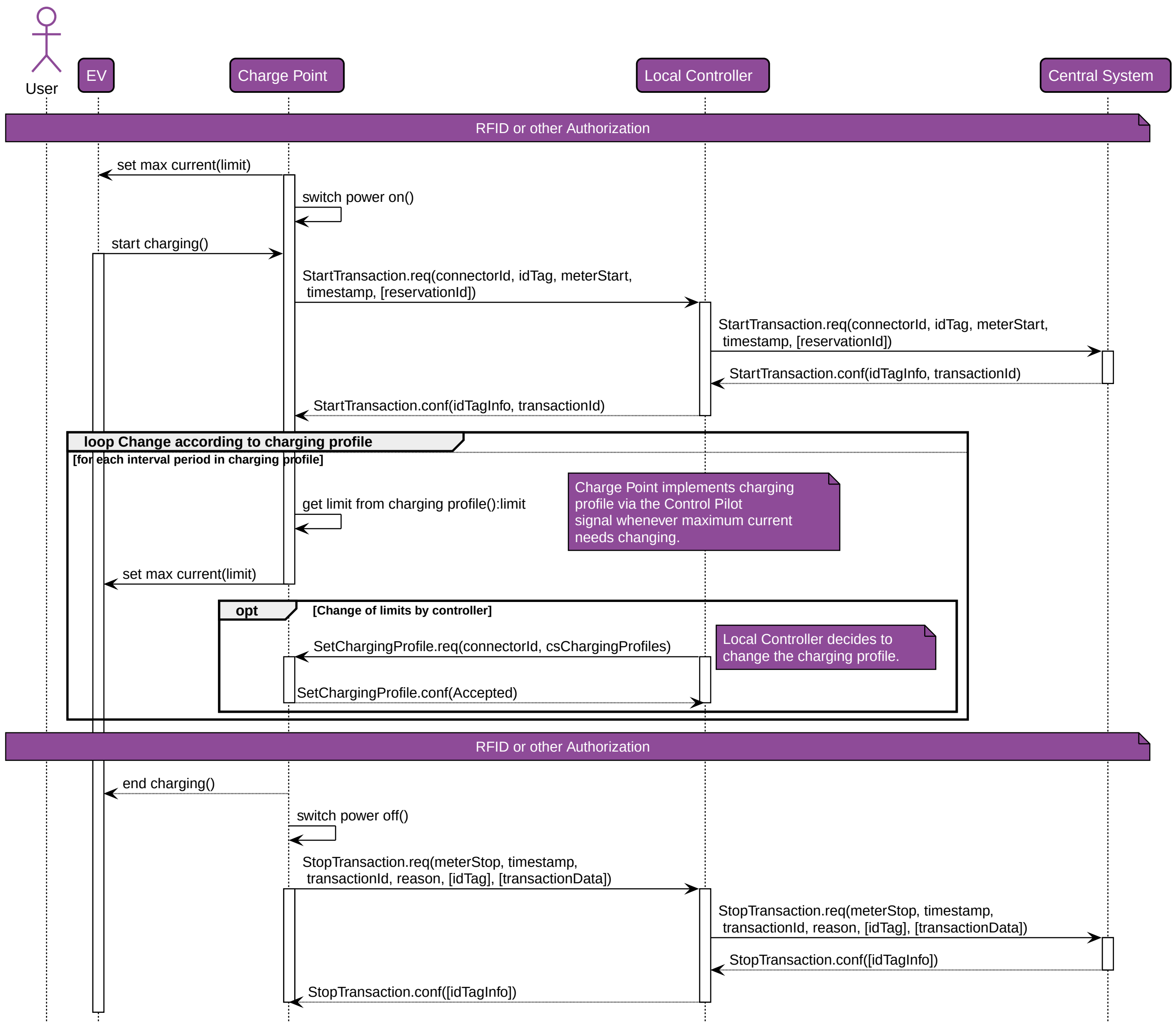
User
EV
Charge Point
Local Controller
Central System
RFID or other Authorization
set max current(limit)
switch power on()
start charging()
StartTransaction.req(connectorId, idTag, meterStart, timestamp, [reservationId])
StartTransaction.req(connectorId, idTag, meterStart, timestamp, [reservationId])
StartTransaction.conf(idTagInfo, transactionId)
StartTransaction.conf(idTagInfo, transactionId)
loop Change according to charging profile
[for each interval period in charging profile]
get limit from charging profile():limit
Charge Point implements charging profile via the Control Pilot signal whenever maximum current needs changing.
set max current(limit)
opt
[Change of limits by controller]
SetChargingProfile.req(connectorId, csChargingProfiles)
Local Controller decides to change the charging profile.
SetChargingProfile.conf(Accepted)
RFID or other Authorization
end charging()
switch power off()
StopTransaction.req(meterStop, timestamp, transactionId, reason, [idTag], [transactionData])
StopTransaction.req(meterStop, timestamp, transactionId, reason, [idTag], [transactionData])
StopTransaction.conf([idTagInfo])
StopTransaction.conf([idTagInfo])

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)**	
	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.

4. Operations Initiated by Charge Point

4.1. Authorize

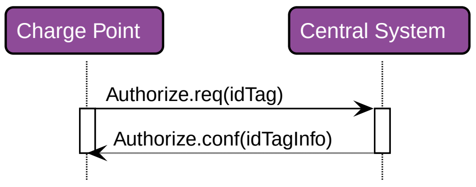

Figure 12. Sequence Diagram: Authorize

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

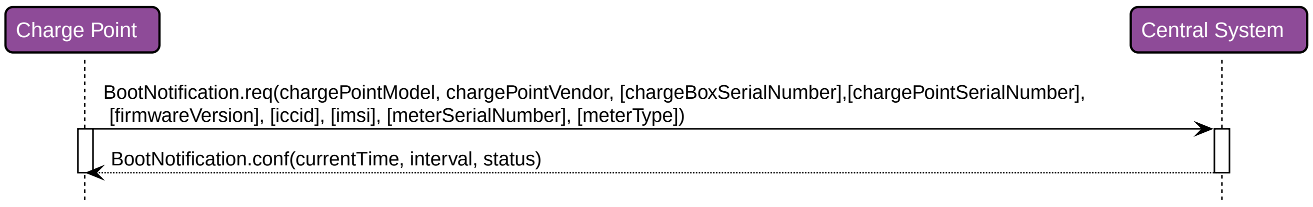
Charge Point
Central System
BootNotification.req(chargePointModel, chargePointVendor, [chargeBoxSerialNumber],[chargePointSerialNumber], [firmwareVersion], [iccid], [imsi], [meterSerialNumber], [meterType])
BootNotification.conf(currentTime, interval, status)

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

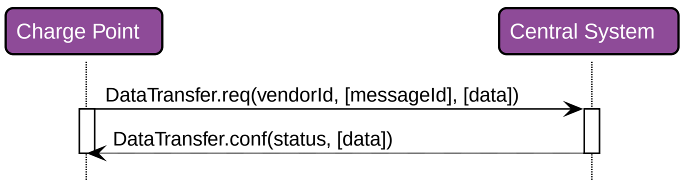
Charge Point
Central System
DataTransfer.req(vendorId, [messageId], [data])
DataTransfer.conf(status, [data])

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

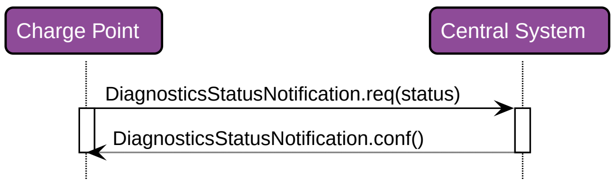
Charge Point
Central System
DiagnosticsStatusNotification.req(status)
DiagnosticsStatusNotification.conf()

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

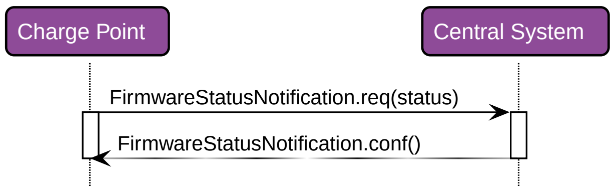
Charge Point
Central System
FirmwareStatusNotification.req(status)
FirmwareStatusNotification.conf()

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

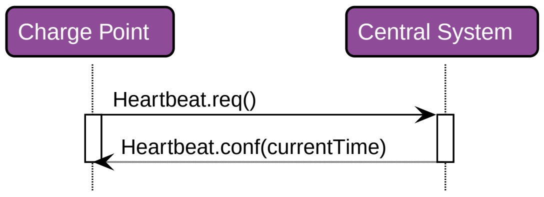

Figure 17. Sequence Diagram: Heartbeat

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

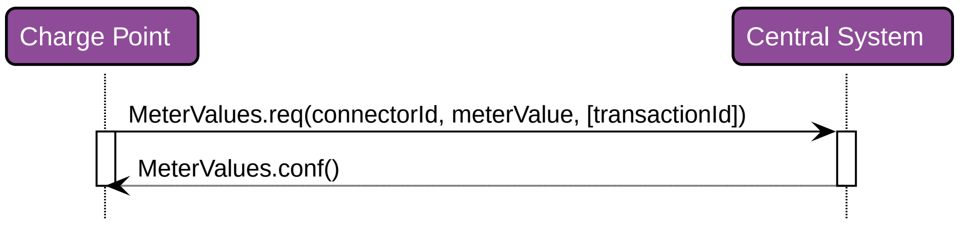

Figure 18. Sequence Diagram: Meter Values

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

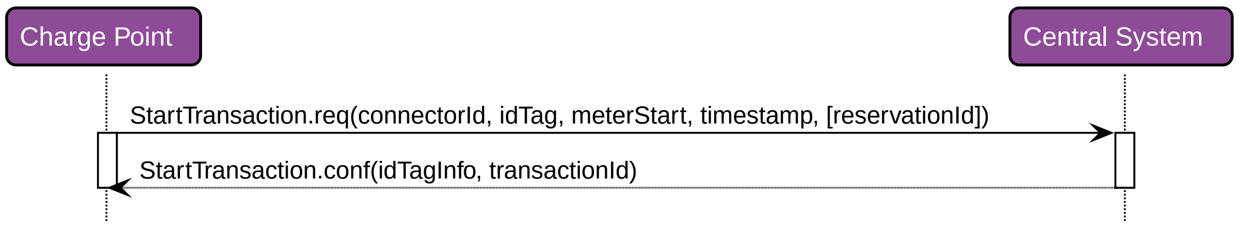
Charge Point
Central System
StartTransaction.req(connectorId, idTag, meterStart, timestamp, [reservationId])
StartTransaction.conf(idTagInfo, transactionId)

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

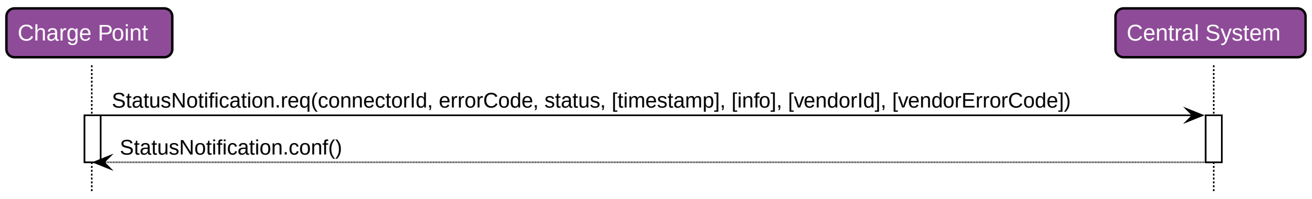
Charge Point
Central System
StatusNotification.req(connectorId, errorCode, status, [timestamp], [info], [vendorId], [vendorErrorCode])
StatusNotification.conf()

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 Prepairing	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*

	DESCRIPTION
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)
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