	DESCRIPTION
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

	DESCRIPTION
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.

ℹ 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

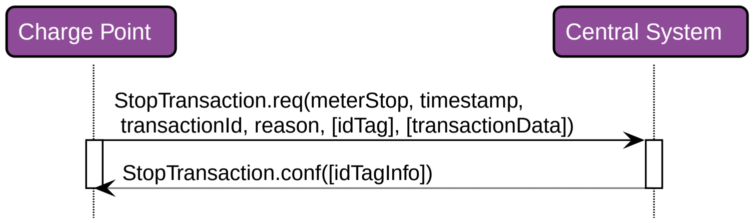

Charge Point
Central System
StopTransaction.req(meterStop, timestamp, transactionId, reason, [idTag], [transactionData])
StopTransaction.conf([idTagInfo])

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.

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.

5. Operations Initiated by Central System

5.1. Cancel Reservation

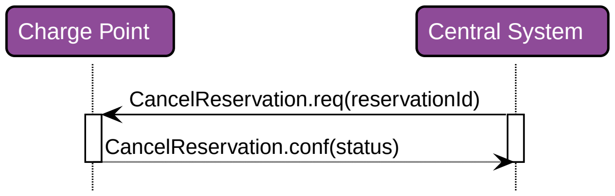

Charge Point
Central System
CancelReservation.req(reservationId)
CancelReservation.conf(status)

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

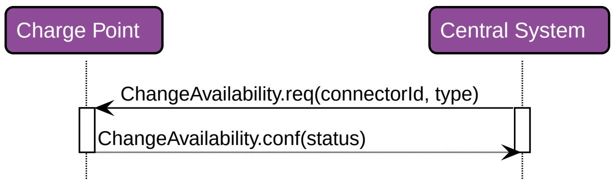

Charge Point
Central System
ChangeAvailability.req(connectorId, type)
ChangeAvailability.conf(status)

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

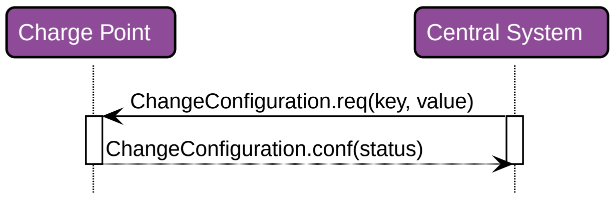

Charge Point
Central System
ChangeConfiguration.req(key, value)
ChangeConfiguration.conf(status)

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

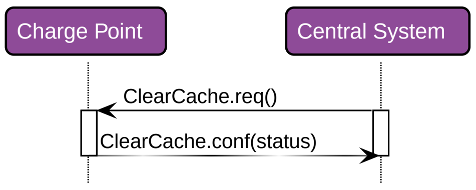


Figure 25. Sequence Diagram: Clear Cache

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

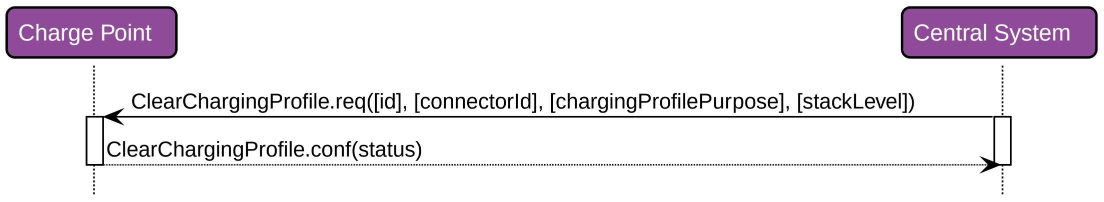

Charge Point
Central System
ClearChargingProfile.req([id], [connectorId], [chargingProfilePurpose], [stackLevel])
ClearChargingProfile.conf(status)

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

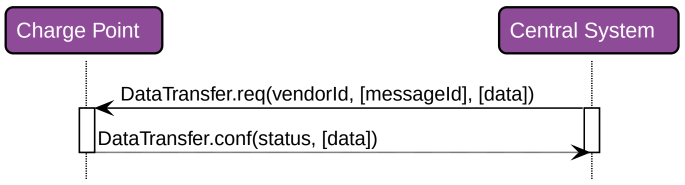

Charge Point
Central System
DataTransfer.req(vendorId, [messageId], [data])
DataTransfer.conf(status, [data])

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

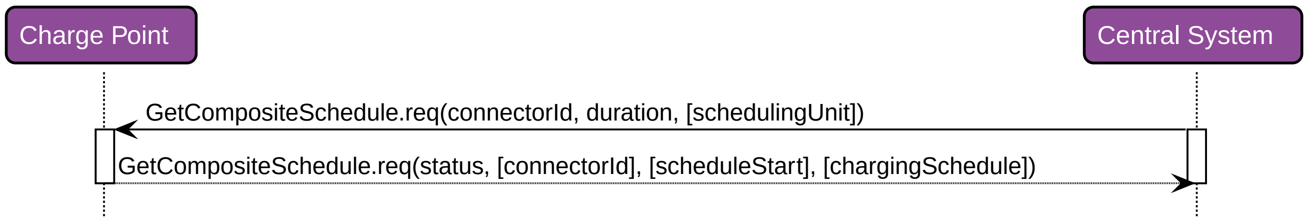

Charge Point
Central System
GetCompositeSchedule.req(connectorId, duration, [schedulingUnit])
GetCompositeSchedule.req(status, [connectorId], [scheduleStart], [chargingSchedule])

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.

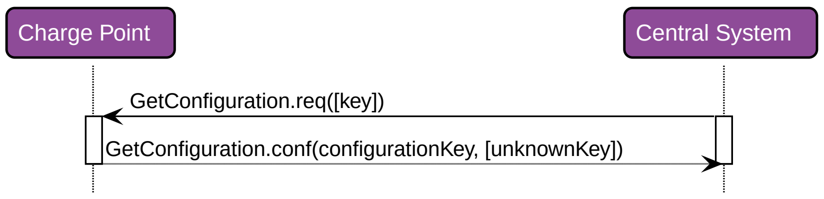
Upon receipt of a GetCompositeSchedule.req, the Charge Point SHALL calculate the Composite Charging Schedule intervals, from the moment the request PDU is received: Time X, up to X + Duration, and send them in the GetCompositeSchedule.conf PDU to the Central System.

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



Charge Point
Central System
GetConfiguration.req([key])
GetConfiguration.conf(configurationKey, [unknownKey])

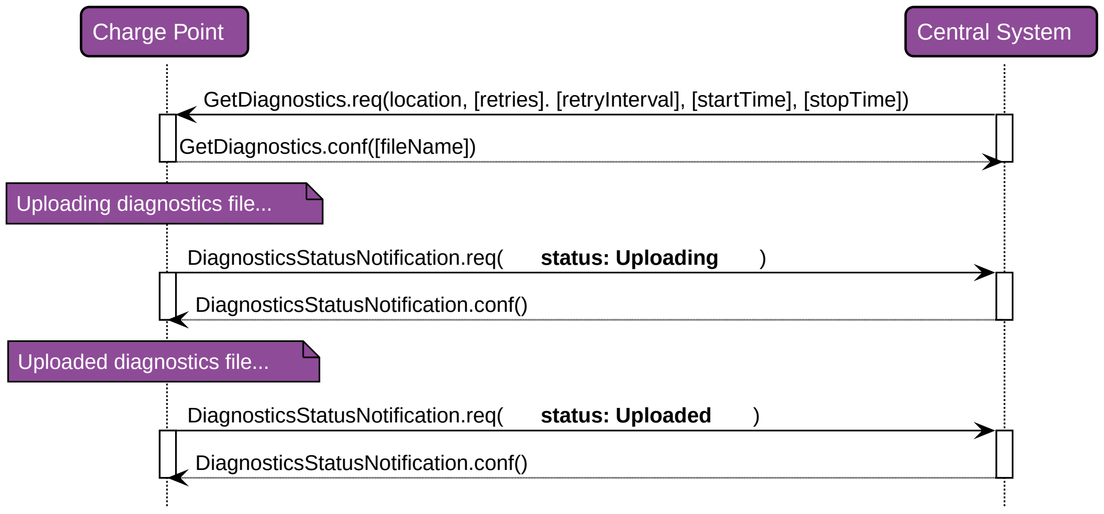
Figure 29. Sequence Diagram: Get Configuration

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



Charge Point
Central System
GetDiagnostics.req(location, [retries]. [retryInterval], [startTime], [stopTime])
GetDiagnostics.conf([fileName])
Uploading diagnostics file...
DiagnosticsStatusNotification.req(status: Uploading)
DiagnosticsStatusNotification.conf()
Uploaded diagnostics file...
DiagnosticsStatusNotification.req(status: Uploaded)
DiagnosticsStatusNotification.conf()

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

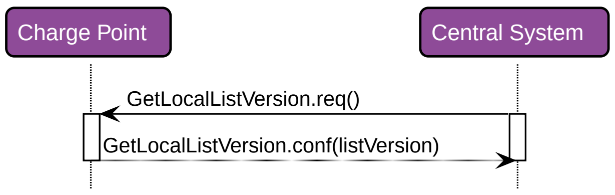

Charge Point
Central System
GetLocalListVersion.req()
GetLocalListVersion.conf(listVersion)

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

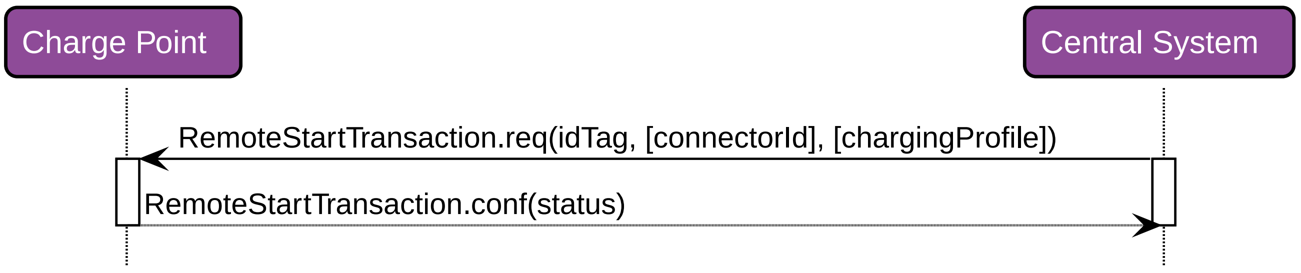

Charge Point
Central System
RemoteStartTransaction.req(idTag, [connectorId], [chargingProfile])
RemoteStartTransaction.conf(status)

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

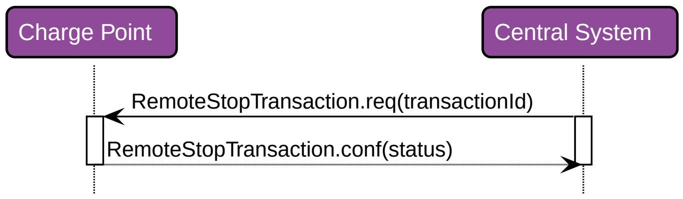

Charge Point
Central System
RemoteStopTransaction.req(transactionId)
RemoteStopTransaction.conf(status)

Figure 33. Sequence Diagram: 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

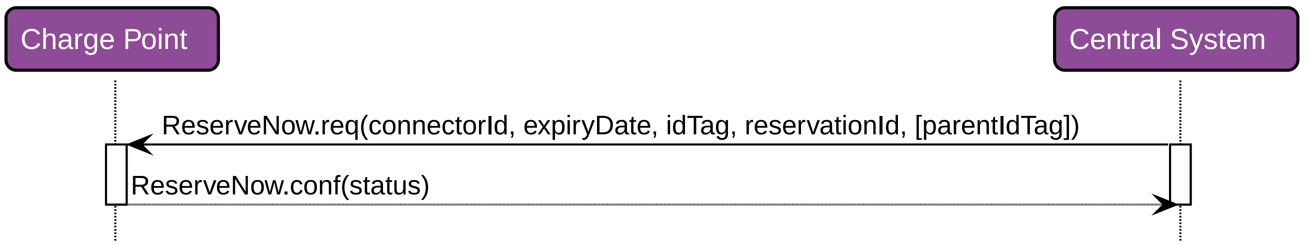

Charge Point
Central System
ReserveNow.req(connectorId, expiryDate, idTag, reservationId, [parentIdTag])
ReserveNow.conf(status)

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

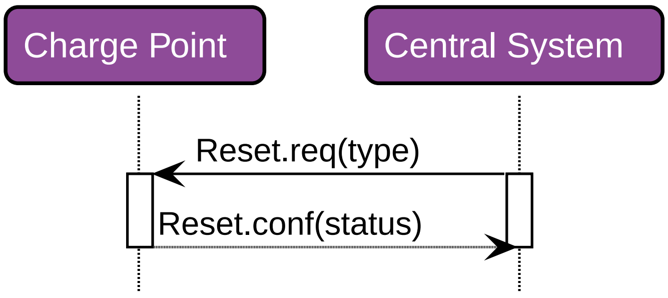


Figure 35. Sequence Diagram: Reset

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.

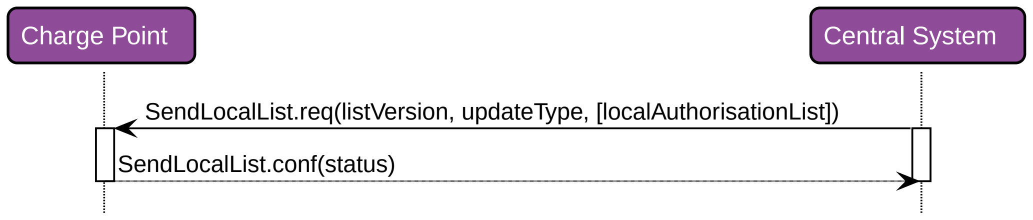
After receipt of a Reset.req, The Charge Point SHALL send a StopTransaction.req for any ongoing transaction before performing the reset. If the Charge Point fails to receive a StopTransaction.conf form the Central System, it shall queue the StopTransaction.req.

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



Charge Point
Central System
SendLocalList.req(listVersion, updateType, [localAuthorisationList])
SendLocalList.conf(status)

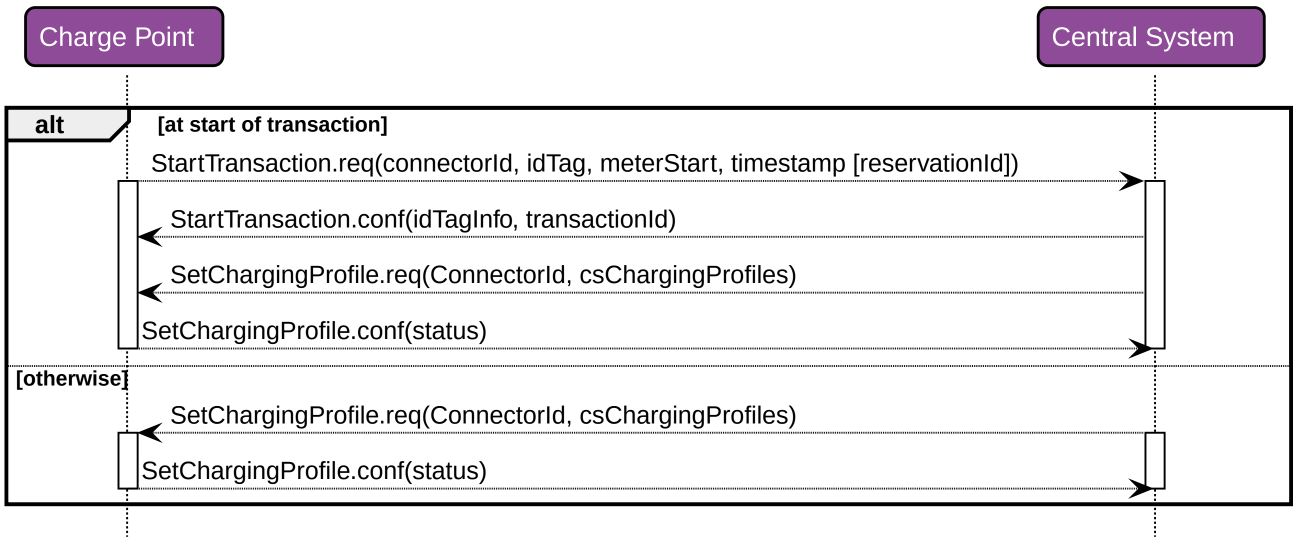
Figure 36. Sequence Diagram: Send Local List

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



Charge Point
Central System
alt
[at start of transaction]
StartTransaction.req(connectorId, idTag, meterStart, timestamp [reservationId])
StartTransaction.conf(idTagInfo, transactionId)
SetChargingProfile.req(ConnectorId, csChargingProfiles)
SetChargingProfile.conf(status)
[otherwise]
SetChargingProfile.req(ConnectorId, csChargingProfiles)
SetChargingProfile.conf(status)

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.

ℹ 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

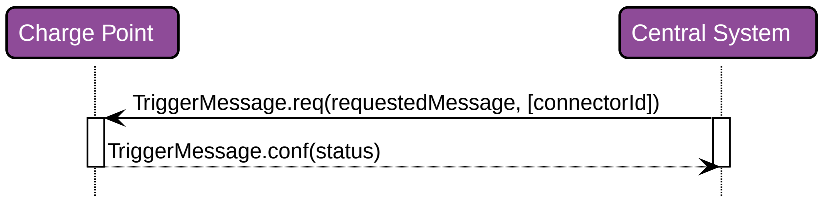

Charge Point
Central System
TriggerMessage.req(requestedMessage, [connectorId])
TriggerMessage.conf(status)

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