Test case document OCTT for OCPP 1.6

FINAL, 2024-04

Table of Contents

Version History

Version	Reviewed by	Modified by	Description
FINAL 2024-04-26	N/a	Open Charge Alliance	Final version

1. Introduction

1.1. About this document

This document is created to describe the test cases that can be executed using the OCPP Compliance Testing Tool (OCTT) for OCPP 1.6.

1.2. Generic conventions

The following conventions / rules apply to all test cases, unless explicitly mentioned otherwise. These will not be mentioned separately at every test case.

- All messages shall comply with the OCPP 1.6 schema’s.
- The messages are to be sent as mentioned in the scenario details except where noted otherwise.
- As an exception to the previous rule, StatusNotification(Charging) and StartTransaction.req may be reversed. This is also the case for StatusNotification(Finishing) and StopTransaction.req.
- Manual actions and actions by external actors will be mentioned in the scenario details between [square brackets].
- When is asked to authenticate by presenting identification, this can be any form of identification. Pressing a start/stop button for example is also allowed in this case.
- Validations will be mentioned and grouped per step.
- Not all test cases need to be passed to have successfully implemented OCPP 1.6. There are test cases which are optional or conditionally optional.
- This document does not specify which tests need to be passed for certification, this will be specified in a separate document.

1.3. General pre- and post- conditions

Unless specifically noted otherwise. the following pre- and post- conditions apply:

- Central System is up and running
- Charge Point is Accepted by the Central System
- Charge Point has a stable active connection to the Central System
- Charge Point connectors are available
- Charge Point is Idle, with no active transactions
- Charge Point is clear of faults
- Charge Point has no charging schedules active
- Charge Point has no active reservations
- Charge Point has no installed local authorization list
- Charge Point has an empty authorization cache
- Charge Point has no more OCPP messages to be sent in queue
- Charge Point is not busy with transfer of diagnostics
- Charge Point is not busy with download of firmware
- Charge Point is not upgrading firmware
- Charge Point is ready to accept/start a charging session
- **MinimumStatusDuration** should be set to *0*. If the Charge Point does not support **MinimumStatusDuration**, the tests are still able pass. The tool will display the 'unexpected' StatusNotification messages in a separate pop-up window. These need to be manually validated by the tester.

2. System Under Test (SUT) Charge Point

This section contains all test cases available in the tool, when configured System Under Test (SUT) Charge Point.

2.1. Cold Boot Charge Point

2.1.1. Cold Boot Charge Point

Table 1. Test Case Id: TC_001_CS

Test case name	**Cold Boot Charge Point**	
Test case Id	TC_001_CS	
Description	This scenario is used to startup the Charge Point and let it register itself at the Central System.	
Purpose	To test if the Charge Point sends the correct messages during the boot process.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Power cycle the Charge Point.] **1.** The Charge Point sends a **BootNotification.req**	**2.** The Central System responds with a **BootNotification.conf**
	3. The Charge Point sends a **BootNotification.req**	**4.** The Central System responds with a **BootNotification.conf**
	[Send a StatusNotification per connector and connectorId=0.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[Every x seconds.] **7.** The Charge Point sends a **Heartbeat.req**	**8.** The Central System responds with a **Heartbeat.conf**
Tool validation(s)	* Step 3: (Message: **BootNotification.req**) *Send BootNotification after interval specified in BootNotification.conf from step 2.* * Step 5: (Message: **StatusNotification.req**) **status** is *Available* * Step 7: (Message: **Heartbeat.req**) *Send a Heartbeat.req every x seconds. x equals* ***interval*** *from step 4.*	* Step 2: (Message: **BootNotification.conf**) The **status** is *Rejected* * Step 4: (Message: **BootNotification.conf**) The **status** is *Accepted* The **interval** is *<Configured Heartbeat interval>*
Expected result(s) / behaviour	n/a	n/a

2.1.2. Cold Boot Charge Point - Pending

Table 2. Test Case Id: TC_002_CS

Test case name	**Cold Boot Charge Point - Pending**
Test case Id	TC_002_CS
Description	This scenario is used to delay the startup for a Charge Point. For example to set the correct configurations.
Purpose	To test if the Charge Point is able to retrieve and set configuration while in pending state.
Prerequisite(s)	n/a

Test case name	**Cold Boot Charge Point - Pending**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Power cycle the Charge Point.] **1.** The Charge Point sends a **BootNotification.req**	**2.** The Central System responds with a **BootNotification.conf**
	4. The Charge Point responds with a **GetConfiguration.conf**	**3.** The Central System sends a **GetConfiguration.req**
	6. The Charge Point responds with a **ChangeConfiguration.conf**	**5.** The Central System sends a **ChangeConfiguration.req**
	7. The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Send a StatusNotification per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
	[Every x seconds.] **11.** The Charge Point sends a **Heartbeat.req**	**12.** The Central System responds with a **Heartbeat.conf**
Tool validation(s)	* Step 6: (Message: **ChangeConfiguration.conf**) **status** is *Accepted* * Step 7: (Message: **BootNotification.req**) *Send BootNotification after interval specified in BootNotification.conf from step 2.* * Step 9: (Message: **StatusNotification.req**) **status** is *Available* * Step 11: (Message: **Heartbeat.req**) *Send a Heartbeat.req every x seconds. x equals **interval** from step 8.*	* Step 2: (Message: **BootNotification.conf**) The **status** is *Pending* * Step 3: (Message: **GetConfiguration.req**) The **key** is *<Omitted>* * Step 5: (Message: **ChangeConfiguration.req**) The **key** is *MeterValueSampleInterval* **value** is *<Configured Meter Value interval>* * Step 8: (Message: **BootNotification.conf**) The **status** is *Accepted* The **interval** is *<Configured Heartbeat interval>*
Expected result(s) / behaviour	n/a	n/a

2.2. Start Charging Session

2.2.1. Regular Charging Session - Plugin First

Table 3. Test Case Id: TC_003_CS

Test case name	**Regular Charging Session - Plugin First**
Test case Id	TC_003_CS
Description	This scenario is used to start a Charging session.
Purpose	To test if the Charge Point is able to start a Charging Session when first doing plugin cable.
Prerequisite(s)	n/a

Test case name	**Regular Charging Session - Plugin First**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver plugs in the cable.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	[EV driver presents identification.] **3.** The Charge Point sends an **Authorize.req**	**4.** The Central System responds with an **Authorize.conf**
	[Step 5 and step 7 may be reversed.] **5.** The Charge Point sends a **StartTransaction.req**	**6.** The Central System responds with a **StartTransaction.conf**
	[Step 5 and step 7 may be reversed.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 7: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 4: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted* * Step 6: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.2.2. Regular Charging Session – Identification First

Table 4. Test Case Id: TC_004_1_CS

Test case name	**Regular Charging Session – Identification First**	
Test case Id	TC_004_1_CS	
Description	This scenario is used to start a Charging session.	
Purpose	To test if the Charge Point is able to start a Charging Session when first doing authorization.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** - Value for "MeterValueSampleInterval" is <Configured Meter Value interval>.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	1. Execute Reusable State *Authorized*	
	2. Manual Action: *EV driver plugs in the cable.*	
	3. The Charge Point sends a **StartTransaction.req**	**4.** The Central System responds with a **StartTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	Note: Step 3 and step 5 may be reversed.	
Tool validation(s)	* Step 5: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 4: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.2.3. Regular Charging Session – Identification First - ConnectionTimeOut

Table 5. Test Case Id: TC_004_2_CS

Test case name	**Regular Charging Session – Identification First - ConnectionTimeOut**	
Test case Id	TC_004_2_CS	
Description	This scenario is used to make a connector available when it is not used.	
Purpose	To test if the Charge Point sets the connector back to *Available*, when the connectionTimeOut is reached.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** - Value for "ConnectionTimeOut" is <Configured connectionTimeout>.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	1. Execute Reusable State *Authorized*	
	[After the configured connectionTimeOut has expired.] **2.** The Charge Point sends a **StatusNotification.req**	**3.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **StatusNotification.req**) **status** is *Available*	n/a
Expected result(s) / behaviour	n/a	n/a

2.3. Stop Charging Session

2.3.1. Stop transaction - IdTag in StopTransaction matches IdTag in StartTransaction

Table 6. Test Case Id: TC_068_CS

Test case name	**Stop transaction - IdTag in StopTransaction matches IdTag in StartTransaction**
Test case Id	TC_068_CS
Description	The Charge Point stops a transaction when a card is swiped with the same idToken as used to start the transaction.
Purpose	Check whether the Charge Point is able to handle a stop transaction with same idToken.
Prerequisite(s)	- If the Charge Point has multiple connectors attached to one RFID reader, then the connector which is NOT under test should be occupied.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): - *Charging*

Test case name	**Stop transaction - IdTag in StopTransaction matches IdTag in StartTransaction**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver authorizes / swipes card with a different IdTag than the one used to start the transaction. This IdTag needs to be configured at the <Configured Valid IdTag 2> field.] **1.** The Charge Point does NOT send an **Authorize.req** and The Charge Point does NOT send a **StopTransaction.req**	
	[EV driver authorizes / swipes card with the IdTag used to start the transaction] [Step 3 and step 5 may be reversed.] **3.** The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	[Step 3 and step 5 may be reversed.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **StopTransaction.req**) The **idTag** matches the **idTag** that was used to start the transaction. * Step 5: (Message: **StatusNotification.req**) The **status** is *Finishing*	n/a
Expected result(s) / behaviour	The Charge Point *only* stops the transaction when receiving the IdTag which was used to start the transaction.	n/a

2.3.2. Stop transaction - ParentIdTag in StopTransaction matches ParentIdTag in StartTransaction

Table 7. Test Case Id: TC_069_CS

Test case name	**Stop transaction - ParentIdTag in StopTransaction matches ParentIdTag in StartTransaction**	
Test case Id	TC_069_CS	
Description	The Charge Point stops a transaction when a card is swiped with the same ParentIdTag as used to start the transaction.	
Purpose	Check whether the Charge Point is able to handle a stop transaction with same ParentIdTag.	
Prerequisite(s)	- If the Charge Point has multiple connectors attached to one RFID reader, then the connector which is NOT under test should be occupied.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver authorizes / swipes card with a different IdTag and the same ParentIdTag than the one used to start the transaction] **1.** The Charge Point sends an **Authorize.req**	**2.** The Central System responds with an **Authorize.conf**
	[Step 3 and step 5 may be reversed.] **3.** The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	[Step 3 and step 5 may be reversed.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**

Test case name	**Stop transaction - ParentIdTag in StopTransaction matches ParentIdTag in StartTransaction**	
Tool validation(s)	* Step 1: (Message: **Authorize.req**) The **idTag** is different from the one used to start the transaction. * Step 3: (Message: **StopTransaction.req**) The **idTag** matches the **idTag** from step 1. * Step 5: (Message: **StatusNotification.req**) The **status** is *Finishing*	* Step 2: (Message: **Authorize.conf**) The **idTagInfo.status** is *Accepted* The **idTagInfo.parentIdTag** matches the **parentIdTag** that was used to start the transaction.
Expected result(s) / behaviour	The Charge Point stops the transaction when receiving a (different) idTag with the same parentIdTag, as the one used to start the transaction.	n/a

2.3.3. EV Side Disconnected - StopTransactionOnEVSideDisconnect = true - UnlockConnectorOnEVSideDisconnect = true

Table 8. Test Case Id: TC_005_1_CS

Test case name	**EV Side Disconnected - StopTransactionOnEVSideDisconnect = true - UnlockConnectorOnEVSideDisconnect = true**	
Test case Id	TC_005_1_CS	
Description	This scenario is used to stop the transaction when the cable is disconnected at EV side.	
Purpose	To test if the Charge Point is able to stop the transaction when the cable is disconnected at EV side and it is configured to do so.	
Prerequisite(s)	- The Charge Point does not have a fixed cable on Charge Point side. - The configuration key *StopTransactionOnEVSideDisconnect* does NOT have the accessibility *ReadOnly* in combination with value *false*.	
Before	**Configuration State(s):** - Value for "MinimumStatusDuration" is "0". - Value for "StopTransactionOnEVSideDisconnect" is "true". - Value for "UnlockConnectorOnEVSideDisconnect" is "true".	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver unplugs cable on EV side.] [Step 1 and step 3 may be reversed.] **1.** The Charge Point sends a **StopTransaction.req**	**2.** The Central System responds with a **StopTransaction.conf**
	[Step 1 and step 3 may be reversed.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StopTransaction.req**) **reason** is *EVDisconnected* * Step 3: (Message: **StatusNotification.req**) **status** is *Finishing*	n/a
Expected result(s) / behaviour	n/a	n/a

2.3.4. EV Side Disconnected - StopTransactionOnEVSideDisconnect = true - UnlockConnectorOnEVSideDisconnect = false

Table 9. Test Case Id: TC_005_2_CS

Test case name	**EV Side Disconnected - StopTransactionOnEVSideDisconnect = true - UnlockConnectorOnEVSideDisconnect = false**	
Test case Id	TC_005_2_CS	
Description	This scenario is used to stop the transaction when the cable is disconnected at EV side.	
Purpose	To test if the Charge Point is able to stop the transaction when the cable is disconnected at EV side and it is configured to do so.	
Prerequisite(s)	- The configuration key *StopTransactionOnEVSideDisconnect* does NOT have the accessibility *ReadOnly* in combination with value *false*.	
Before	**Configuration State(s):** - Value for "MinimumStatusDuration" is "0". - Value for "StopTransactionOnEVSideDisconnect" is "true". - Value for "UnlockConnectorOnEVSideDisconnect" is "false".	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver unplugs cable on EV side.] [Step 1 and step 3 may be reversed.] **1.** The Charge Point sends a **StopTransaction.req**	**2.** The Central System responds with a **StopTransaction.conf**
	[Step 1 and step 3 may be reversed.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	6. The Charge Point responds with a **UnlockConnector.conf**	**5.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 1: (Message: **StopTransaction.req**) **reason** is *EVDisconnected* * Step 3: (Message: **StatusNotification.req**) **status** is *Finishing* OR *Available*	* Step 6: (Message: **UnlockConnector.conf**) **status** is *Unlocked* OR *NotSupported*
Expected result(s) / behaviour	n/a	n/a

2.3.5. EV Side Disconnected - StopTransactionOnEVSideDisconnect = false - UnlockConnectorOnEVSideDisconnect = false

Table 10. Test Case Id: TC_005_3_CS

Test case name	**EV Side Disconnected - StopTransactionOnEVSideDisconnect = false - UnlockConnectorOnEVSideDisconnect = false**
Test case Id	TC_005_3_CS
Description	This scenario is used to keep the transaction active, even when the cable is disconnected at EV side.
Purpose	To test if the Charge Point is able to keep the transaction active, when the cable is disconnected at EV side and the Charge Point is configured to do so.
Prerequisite(s)	- The configuration key *StopTransactionOnEVSideDisconnect* is implemented AND has the accessibility *ReadWrite*.

Test case name	EV Side Disconnected - StopTransactionOnEVSideDisconnect = false - UnlockConnectorOnEVSideDisconnect = false	
Before	**Configuration State(s):** - Value for "MinimumStatusDuration" is "0". - Value for "StopTransactionOnEVSideDisconnect" is "false". - Value for "UnlockConnectorOnEVSideDisconnect" is "false".	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver unplugs cable on EV side.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	4. The Charge Point responds with a **RemoteStopTransaction.conf**	**3.** The Central System sends a **RemoteStopTransaction.req**
	[Step 5 and step 7 may be reversed.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[Step 5 and step 7 may be reversed.] **7.** The Charge Point sends a **StopTransaction.req**	**8.** The Central System responds with a **StopTransaction.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *SuspendedEV* AND/OR *SuspendedEVSE* **info** is *EV side disconnected* * Step 5: (Message: **StatusNotification.req**) **status** is *Finishing* (OR *Available* in case of a fixed cable) * Step 7: (Message: **StopTransaction.req**) **reason** is *Remote*	n/a
Expected result(s) / behaviour	n/a	n/a

2.4. Cache

2.4.1. Regular Start Charging Session – Cached Id

Table 11. Test Case Id: TC_007_CS

Test case name	Regular Start Charging Session – Cached Id
Test case Id	TC_007_CS
Description	This scenario is used to start a transaction with an id stored in the Authorization cache.
Purpose	To test if the Charge Point is able to start a transaction with an id which is stored in the Authorization cache.
Prerequisite(s)	The Charge Point has a cache AND The configuration key *AuthorizeRemoteTxRequests* does not have an accessibility of *ReadOnly* in combination with the value *false*.

Test case name	**Regular Start Charging Session – Cached Id**	
Before	**Configuration State(s):** - *AuthorizationCacheEnabled* is *true*. - *AuthorizeRemoteTxRequests* is *true*. (If implemented) - *LocalPreAuthorize* is *true*.	
	Memory State(s): - *IdTagCached* for <Configured valid IdTag>	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver plugs in the cable] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	4. The Charge Point responds with a **RemoteStartTransaction.conf**	**3.** The Central System sends a **RemoteStartTransaction.req**
	[Steps 5 and 7 may be reversed] **5.** The Charge Point sends a **StartTransaction.req**	**6.** The Central System responds with a **StartTransaction.conf**
	7. The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 4: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 7: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 6: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.4.2. Clear Authorization Data in Authorization Cache

Table 12. Test Case Id: TC_061_CS

Test case name	**Clear Authorization Data in Authorization Cache**
Test case Id	TC_061_CS
Description	The Central System can clear the Authorization Cache of a Charge Point.
Purpose	Check whether the Charge Point can handle the message to clear the Authorization Cache.
Prerequisite(s)	- The Charge Point has an authorization cache implemented.
Before	**Configuration State(s):** - Value for "AuthorizationCacheEnabled" is "true". - Value for "LocalPreAuthorize" is "true". - Value for "ConnectionTimeOut" is <Configured ConnectionTimeout>.
	Memory State(s): n/a
	Reusable State(s): - *Authorized*

Test case name	**Clear Authorization Data in Authorization Cache**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	1. The Charge Point sends an **StatusNotification.req** to the Central System.	**2.** The Central system responds with an **StatusNotification.conf**
	[EV driver authorizes / swipes valid card OR wait for the connectionTimeout if the Charge Point does not deauthorize the transaction after swiping again.] **3.** The Charge Point sends an **StatusNotification.req** to the Central System.	**4.** The Central system responds with an **StatusNotification.conf**
	6. The Charge Point responds with a **ClearCache.conf**	**5.** The Central System sends a **ClearCache.req**
	[The EV driver authorizes / swipes card with the authorization token which was used at step 1.] **7.** The Charge Point sends an **Authorize.req**	**8.** The Central System responds with an **Authorize.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 3: (Message: **StatusNotification.req**) **status** is *Available* * Step 6: (Message: **ClearCache.conf**) **status** is *Accepted*	* Step 8: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	The Charge Point Authorization Cache is cleared.	The Central System is able to send a message to clear the cache.

2.5. Core Profile - Remote actions Happy flow

2.5.1. Remote Start Charging Session – Cable Plugged in First

Table 13. Test Case Id: TC_010_CS

Test case name	**Remote Start Charging Session – Cable Plugged in First**	
Test case Id	TC_010_CS	
Description	This scenario is used to start a transaction remotely.	
Purpose	To test if the Charge point is able to start a transaction after receiving a RemoteStartTransaction.req from the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	- *Charging*	
Tool validation(s)	n/a	n/a
Expected result(s) / behaviour	n/a	n/a

2.5.2. Remote Start Charging Session – Remote Start First

Table 14. Test Case Id: TC_011_1_CS

Test case name	**Remote Start Charging Session – Remote Start First**	
Test case Id	TC_011_1_CS	
Description	This scenario is used to start a transaction remotely.	
Purpose	To test if the Charge point is able to start a transaction after receiving a RemoteStartTransaction.req from the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetConfiguration.conf**	**1.** The Central System sends a **GetConfiguration.req**
	4. The Charge Point responds with a **RemoteStartTransaction.conf**	**3.** The Central System sends a **RemoteStartTransaction.req**
	[If AuthorizeRemoteTxRequests = true (from step 2), send an Authorize.req.] **5.** The Charge Point sends an **Authorize.req**	**6.** The Central System responds with an **Authorize.conf**
	7. The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
	[EV driver plugs in the cable.] **9.** The Charge Point sends a **StartTransaction.req**	**10.** The Central System responds with a **StartTransaction.conf**
	11. The Charge Point sends a **StatusNotification.req**	**12.** The Central System responds with a **StatusNotification.conf**

Test case name	Remote Start Charging Session – Remote Start First	
Tool validation(s)	* Step 2: (Message: **GetConfiguration.conf**) The **configurationKey.key** is *AuthorizeRemoteTxRequests* * Step 4: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 7: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 11: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 1: (Message: **GetConfiguration.req**) The **key** is *AuthorizeRemoteTxRequests* * Step 6: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted* * Step 10: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.5.3. Remote Start Charging Session – Time Out

Table 15. Test Case Id: TC_011_2_CS

Test case name	Remote Start Charging Session – Time Out	
Test case Id	TC_011_2_CS	
Description	This scenario is used to set a connector back to available, after receiving a RemoteStartTransaction.req and it takes to long to plugin the cable.	
Purpose	To test if the Charge Point sets the connector back to available, after reaching the configured connection timeout.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** - Value of "ConnectionTimeOut" is <Configured connectionTimeout>.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetConfiguration.conf**	**1.** The Central System sends a **GetConfiguration.req**
	4. The Charge Point responds with a **RemoteStartTransaction.conf**	**3.** The Central System sends a **RemoteStartTransaction.req**
	[If AuthorizeRemoteTxRequests = true (from step 2), send an Authorize.req.] **5.** The Charge Point sends an **Authorize.req**	**6.** The Central System responds with an **Authorize.conf**
	7. The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
	[After the configured connection timeout has been reached.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**

Test case name	**Remote Start Charging Session – Time Out**	
Tool validation(s)	* Step 2: (Message: **GetConfiguration.conf**) The **configurationKey.key** is *AuthorizeRemoteTxRequests* * Step 4: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 7: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 9: (Message: **StatusNotification.req**) **status** is *Available*	* Step 1: (Message: **GetConfiguration.req**) The **key** is *AuthorizeRemoteTxRequests* * Step 6: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.5.4. Remote Stop Charging Session

Table 16. Test Case Id: TC_012_CS

Test case name	**Remote Stop Charging Session**	
Test case Id	TC_012_CS	
Description	This scenario is used to remotely stop a transaction.	
Purpose	To test if the Charge Point will stop a transaction, when requested by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **RemoteStopTransaction.conf**	**1.** The Central System sends a **RemoteStopTransaction.req**
	[Steps 3 and 5 may be reversed] **3.** The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **RemoteStopTransaction.conf**) **status** is *Accepted* * Step 3: (Message: **StopTransaction.req**) **reason** is *Remote* * Step 5: (Message: **StatusNotification.req**) **status** is *Finishing*	
Expected result(s) / behaviour	n/a	n/a

2.6. Core Profile - Resetting Happy Flow

2.6.1. Hard Reset Without transaction

Table 17. Test Case Id: TC_013_CS

Test case name	**Hard Reset Without transaction**	
Test case Id	TC_013_CS	
Description	This scenario is used to hard reset a Charge Point, while no transaction is active.	
Purpose	To test if the Charge Point will hard reset, after being requested by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeAvailability.conf**	**1.** The Central System sends a **ChangeAvailability.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	6. The Charge Point responds with a **Reset.conf**	**5.** The Central System sends a **Reset.req**
	7. The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
	12. The Charge Point responds with a **ChangeAvailability.conf**	**11.** The Central System sends a **ChangeAvailability.req**
	13. The Charge Point sends a **StatusNotification.req**	**14.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **ChangeAvailability.conf**) The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) **connectorId** is *<Configured ConnectorId>* **status** is *Unavailable* * Step 6: (Message: **Reset.conf**) **status** is *Accepted* * Step 9: (Message: **StatusNotification.req**) **connectorId** is *<Configured ConnectorId>* **status** is *Unavailable* (Message: **StatusNotification.req**) *The other StatusNotification messages.* **status** is *Available* * Step 12: (Message: **ChangeAvailability.conf**) The **status** is *Accepted* * Step 13: (Message: **StatusNotification.req**) **connectorId** is *<Configured ConnectorId>* **status** is *Available*	* Step 1: (Message: **ChangeAvailability.req**) The **connectorId** is *<Configured ConnectorId>* The **type** is *Inoperative* * Step 5: (Message: **Reset.req**) The **type** is *Hard* * Step 8: (Message: **BootNotification.conf**) **status** is *Accepted* * Step 11: (Message: **ChangeAvailability.req**) The **connectorId** is *<Configured ConnectorId>* The **type** is *Operative*

Test case name	**Hard Reset Without transaction**	
Expected result(s) / behaviour	n/a	n/a

2.6.2. Soft Reset Without Transaction

Table 18. Test Case Id: TC_014_CS

Test case name	**Soft Reset Without Transaction**	
Test case Id	TC_014_CS	
Description	This scenario is used to soft reset a Charge Point, while no transaction is active.	
Purpose	To test if the Charge Point will soft reset, after being requested by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeAvailability.conf**	**1.** The Central System sends a **ChangeAvailability.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	6. The Charge Point responds with a **Reset.conf**	**5.** The Central System sends a **Reset.req**
	[This message is optional.] **7.** The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[These StatusNotification messages will only be sent if step 7 is sent.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
	12. The Charge Point responds with a **ChangeAvailability.conf**	**11.** The Central System sends a **ChangeAvailability.req**
	13. The Charge Point sends a **StatusNotification.req**	**14.** The Central System responds with a **StatusNotification.conf**

Test case name	**Soft Reset Without Transaction**	
Tool validation(s)	* Step 2: (Message: **ChangeAvailability.conf**) The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) **connectorId** is *<Configured ConnectorId>* **status** is *Unavailable* * Step 6: (Message: **Reset.conf**) **status** is *Accepted* * Step 9: (Message: **StatusNotification.req**) **connectorId** is *<Configured ConnectorId>* **status** is *Unavailable* (Message: **StatusNotification.req**) *The other StatusNotification messages.* **status** is *Available* * Step 12: (Message: **ChangeAvailability.conf**) The **status** is *Accepted* * Step 13: (Message: **StatusNotification.req**) **connectorId** is *<Configured ConnectorId>* **status** is *Available*	* Step 1: (Message: **ChangeAvailability.req**) The **connectorId** is *<Configured ConnectorId>* The **type** is *Inoperative* * Step 5: (Message: **Reset.req**) The **type** is *Soft* * Step 8: (Message: **BootNotification.conf**) **status** is *Accepted* * Step 11: (Message: **ChangeAvailability.req**) The **connectorId** is *<Configured ConnectorId>* The **type** is *Operative*
Expected result(s) / behaviour	n/a	n/a

2.6.3. Hard Reset With Transaction

Table 19. Test Case Id: TC_015_CS

Test case name	**Hard Reset With Transaction**	
Test case Id	TC_015_CS	
Description	This scenario is used to hard reset a Charge Point, while a transaction is active.	
Purpose	To test if the Charge Point will hard reset, after being requested by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **Reset.conf**	**1.** The Central System sends a **Reset.req**
	[Needs to be sent either before or after step 7.] **3.** The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	[Needs to be sent if step 3 is sent. Otherwise it is optional.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	7. The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**

Test case name	Hard Reset With Transaction	
Tool validation(s)	* Step 2: (Message: **Reset.conf**) **status** is *Accepted* * Step 3: (Message: **StopTransaction.req**) **reason** is *HardReset* * Step 5: (Message: **StatusNotification.req**) **status** is *Finishing* * Step 9: (Message: **StatusNotification.req**) **connectorId** is *<The connector which had the ongoing transaction>* **status** is *Finishing OR Preparing* (Message: **StatusNotification.req**) *The other StatusNotification messages.* **status** is *Available*	* Step 1: (Message: **Reset.req**) The **type** is *Hard* * Step 8: (Message: **BootNotification.conf**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.6.4. Soft Reset With Transaction

Table 20. Test Case Id: TC_016_CS

Test case name	Soft Reset With Transaction	
Test case Id	TC_016_CS	
Description	This scenario is used to soft reset a Charge Point, while a transaction is active.	
Purpose	To test if the Charge Point will soft reset, after being requested by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **Reset.conf**	**1.** The Central System sends a **Reset.req**
	3. The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[This message is sent optionally.] **7.** The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Only send if step 7 is sent.] [Send per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**

Test case name	**Soft Reset With Transaction**	
Tool validation(s)	* Step 2: (Message: **Reset.conf**) **status** is *Accepted* * Step 3: (Message: **StopTransaction.req**) **reason** is *SoftReset* * Step 5: (Message: **StatusNotification.req**) **status** is *Finishing* * Step 9: (Message: **StatusNotification.req**) **connectorId** is *<The connector which had the ongoing transaction>* **status** is *Finishing OR Preparing* (Message: **StatusNotification.req**) *The other StatusNotification messages.* **status** is *Available*	* Step 1: (Message: **Reset.req**) The **type** is *Soft* * Step 8: (Message: **BootNotification.conf**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.7. Core Profile - Unlocking Happy flow

2.7.1. Unlock connector - no charging session running (Not fixed cable)

Table 21. Test Case Id: TC_017_1_CS

Test case name	**Unlock connector - no charging session running (Not fixed cable)**	
Test case Id	TC_017_1_CS	
Description	This scenario is used to unlock a connector of a Charge Point.	
Purpose	To test if the Charge Point unlocks the connector, when requested by the Central System.	
Prerequisite(s)	Charging Station does not have a fixed cable.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *Unlocked*	n/a
Expected result(s) / behaviour	n/a	n/a

2.7.2. Unlock connector - no charging session running (Fixed cable)

Table 22. Test Case Id: TC_017_2_CS

Test case name	**Unlock connector - no charging session running (Fixed cable)**	
Test case Id	TC_017_2_CS	
Description	This scenario describes how to Charge Point should react to an UnlockConnector.req, when having a fixed cable.	
Purpose	To test if the Charge Point is able to notify the Central System it does not support the unlocking of a connector.	
Prerequisite(s)	Charging Station has a fixed cable.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *NotSupported*	n/a
Expected result(s) / behaviour	n/a	n/a

2.7.3. Unlock Connector - With Charging Session

Table 23. Test Case Id: TC_018_1_CS

Test case name	**Unlock Connector - With Charging Session (Not fixed cable)**	
Test case Id	TC_018_1_CS	
Description	This scenario is used to unlock a connector of a Charge Point, while a transaction is ongoing.	
Purpose	To test if the Charge Point unlocks the connector, when requested by the Central System.	
Prerequisite(s)	Charging Station does not have a fixed cable.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
	3. The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[EV driver unplugs the cable.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *Unlocked* * Step 3: (Message: **StopTransaction.req**) **reason** is *UnlockCommand* * Step 5: (Message: **StatusNotification.req**) **status** is *Finishing* * Step 7: (Message: **StatusNotification.req**) **status** is *Available*	n/a
Expected result(s) / behaviour	n/a	n/a

2.7.4. Unlock Connector - With Charging Session

Table 24. Test Case Id: TC_018_2_CS

Test case name	**Unlock Connector - With Charging Session (Fixed cable)**
Test case Id	TC_018_2_CS
Description	This scenario describes how to Charge Point should react to an UnlockConnector.req, when having a fixed cable and an ongoing transaction.
Purpose	To test if the Charge Point is able to notify the Central System it does not support the unlocking of a connector.
Prerequisite(s)	Charging Station has a fixed cable.

Test case name	**Unlock Connector - With Charging Session (Fixed cable)**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *NotSupported*	n/a
Expected result(s) / behaviour	n/a	n/a

2.8. Core Profile - Configuration Happy flow

2.8.1. Retrieve configuration

Table 25. Test Case Id: TC_019_CS

Test case name	**Retrieve configuration**	
Test case Id	TC_019_CS	
Description	The Central System is able to retrieve all available or specific configuration keys.	
Purpose	To check whether the Charge Point has all required keys configured.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
	Charge Point (SUT)	**Central System (Tool)**
Scenario Detail(s)	**2.** The Charge Point responds with a **GetConfiguration.conf**.	**1.** The Central Systems sends a **GetConfiguration.req** message to the Charge Point.
	4. The Charge Point responds with a **GetConfiguration.conf**.	**3.** The Central Systems sends a **GetConfiguration.req** message to the Charge Point.
	6. The Charge Point responds with a **GetConfiguration.conf**.	**5.** The Central Systems sends a **GetConfiguration.req** message to the Charge Point.
	8. The Charge Point responds with a **GetConfiguration.conf**.	**7.** The Central Systems sends a **GetConfiguration.req** message to the Charge Point.

Test case name	**Retrieve configuration**	
Tool validation(s)	* Step 2: (Message: **GetConfiguration.conf**) **unknownKey** list is *<Empty>* **configurationKey.key** should be *SupportedFeatureProfiles* * Step 4: (Message: **GetConfiguration.conf**) - Contains at least all required keys from the supported profiles from step 2. - Check if **accessibility** contains the correct value. **Core:** **Configuration Key** / **accessibility** AuthorizeRemoteTxRequests / R OR RW ClockAlignedDataInterval / RW ConnectionTimeOut / RW ConnectorPhaseRotation / RW GetConfigurationMaxKeys / R HeartbeatInterval / RW LocalAuthorizeOffline / RW LocalPreAuthorize / RW MeterValuesAlignedData / RW MeterValuesSampledData / RW MeterValueSampleInterval / RW NumberOfConnectors / R ResetRetries / RW StopTransactionOnInvalidId / RW StopTxnAlignedData / RW StopTxnSampledData / RW SupportedFeatureProfiles / R TransactionMessageAttempts / RW TransactionMessageRetryInterval / RW UnlockConnectorOnEVSideDisconnect / R OR RW *The required configuration keys for the optional Feature Profiles need to be validated manually for now. Later this will be included to the validation of the OCTT.* **Local Auth List Management:** LocalAuthListEnabled / RW LocalAuthListMaxLength / R SendLocalListMaxLength / R **Smart Charging Profile:** ChargeProfileMaxStackLevel / R ChargingScheduleAllowedChargingRateUnit / R ChargingScheduleMaxPeriods / R MaxChargingProfilesInstalled / R **Reservation:** *None* **Remote Trigger:** *None* * Step 8: (Message: **GetConfiguration.conf**) **unknownKey** list is *<Empty>* **configurationKey** list contains all the keys requested in step 7.	* Step 1: (Message: **GetConfiguration.req**) The **key** is *SupportedFeatureProfiles* * Step 3: (Message: **GetConfiguration.req**) The **key** is *<Empty>* * Step 5: (Message: **GetConfiguration.req**) The **key** is *GetConfigurationMaxKeys* * Step 7: (Message: **GetConfiguration.req**) - Contains a list of configuration keys, that consists of keys picked from the list returned in step 4. - The length of the list equals the value of GetConfigurationMaxKeys returned in step 6 or the length of the list returned in step 4, whichever is less.

Test case name	**Retrieve configuration**	
Expected result(s) / behaviour	All required keys are configured.	The Central System is able to retrieve the values of all requested configuration keys.

2.8.2. Change/set Configuration

Table 26. Test Case Id: TC_021_CS

Test case name	**Change/set Configuration**	
Test case Id	TC_021_CS	
Description	This scenario is used to set the value of a configuration key.	
Purpose	To test if the Charge Point sets the configuration key value, specified by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** The value of "MeterValueSampleInterval" is NOT <Configured Meter Value interval>.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
	4. The Charge Point responds with a **GetConfiguration.conf**	**3.** The Central System sends a **GetConfiguration.req**
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) **status** is *Accepted* * Step 4: (Message: **GetConfiguration.conf**) **configurationKey.key** is *MeterValueSampleInterval* **configurationKey.value** is *<Configured Meter Value interval>*	* Step 1: (Message: **ChangeConfiguration.req**) The **key** is *MeterValueSampleInterval* The **value** is *<Configured Meter Value interval>* * Step 3: (Message: **GetConfiguration.req**) The **key** is *MeterValueSampleInterval*
Expected result(s) / behaviour	n/a	n/a

2.9. Meter values

2.9.1. Sampled Meter Values

Table 27. Test Case Id: TC_070_CS

Test case name	**Sampled Meter Values**
Test case Id	TC_070_CS
Description	The Charge Point is able to send different kinds of Sampled MeterValues with a certain interval. What MeterValues are to be sent and at what time(intervals) is configurable.
Purpose	Check whether the Charge Point is able to send MeterValues as configured.
Prerequisite(s)	n/a

Test case name	Sampled Meter Values	
Before	**Configuration State(s):** - *MeterValueSampleInterval* is *<Configured Meter Value Sample interval>*. - *ClockAlignedDataInterval* is *0*.	
	Memory State(s): n/a	
	Reusable State(s): - *GetConfiguration* for key *MeterValuesSampledData* - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Every x seconds after starting the transaction as configured by the Configuration Key **MeterValueSampleInterval**.] **1.** The Charge Point sends a **MeterValues.req** to the Central System.	**2.** The Central System responds with a **MeterValues.conf** to the Charge Point.
	[Three times the configured MeterValueSampleInterval (in seconds) after starting the transaction.]	n/a
Tool validation(s)	* Step 3: (Message: **MeterValues.req**) - Between the **MeterValue.timestamp** fields of the sent MeterValues.req should be an interval of x seconds. - The **sampledValue.context** should be *Sample.Periodic* - The **sampledValue** list should contain an sampledValue for each **sampledValue.measurand** configured in the **MeterValuesSampledData** Configuration Key (the measurands returned in step 2). - When the value for *MeterValuesSampledData* is empty the measurand *Energy.Active.Import.Register* is assumed as default.	**Expected result(s) / behaviour**

2.9.2. Clock-aligned Meter values

Table 28. Test Case Id: TC_071_CS

Test case name	Clock-aligned Meter Values
Test case Id	TC_071_CS
Description	The Charge Point is able to send different kinds of Clock-aligned MeterValues with a certain interval. What MeterValues are to be sent and at what time(intervals) is configurable.
Purpose	Check whether the Charge Point is able to send MeterValues as configured.
Prerequisite(s)	n/a
Before	**Configuration State(s):** - *MeterValueSampleInterval* is *0*. - *ClockAlignedDataInterval* is *<Configured Clock Aligned Data interval>*.
	Memory State(s): n/a
	Reusable State(s): - *GetConfiguration* for key *MeterValuesAlignedData* - *Charging*

Test case name	**Clock-aligned Meter Values**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Will be sent every x seconds as configured in the Configuration Key **ClockAlignedDataInterval**).] **1.** The Charge Point sends a **MeterValues.req** to the Central System.	**2.** The Central System responds with a **MeterValues.conf** to the Charge Point.
	[Three times the configured ClockAlignedDataInterval (in seconds) after starting the transaction.]	n/a
Tool validation(s)	* Step 3: (Message: **MeterValues.req**) - Between the **MeterValue.timestamp** fields of the sent MeterValues.req should be an interval of x seconds as configured with Configuration Key **ClockAlignedDataInterval**. - The **MeterValue.timestamp** should contain a Clock-aligned value. (For example in case of a 20s interval, the seconds should be of value; 0, 20, 40) - The **sampledValue.context** should be *Sample.Clock* - The **sampledValue** list should contain an sampledValue for each **sampledValue.measurand** configured in the **MeterValuesAlignedData** Configuration Key (the measurands returned in step 2). - When the value for *MeterValuesAlignedData* is empty the measurand *Energy.Active.Import.Register* is assumed as default.	n/a
Expected result(s) / behaviour	n/a	n/a

2.10. Core Profile - Basic Actions Non-happy flow

2.10.1. Start Charging Session – Authorize invalid

Table 29. Test Case Id: TC_023_CS

Test case name	**Start Charging Session – Authorize invalid**	
Test case Id	TC_023_CS	
Description	This scenario is used to inform the Charge Point that the EV Driver is not Authorized to start a transaction.	
Purpose	To test if the Charge Point does not start a transaction after Authorization fails.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** - Value for "MinimumStatusDuration" is "10". - Value for "LocalPreAuthorize" is "true".	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver plugs in the cable.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	[EV driver presents invalid identification.] **3.** The Charge Point sends an **Authorize.req**	**4.** The Central System responds with an **Authorize.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing*	* Step 4: (Message: **Authorize.conf**) **idTagInfo.status** is *Invalid*
Expected result(s) / behaviour	The Charge Point does NOT start a transaction.	n/a

2.10.2. Start Charging Session Lock Failure

Table 30. Test Case Id: TC_024_CS

Test case name	**Start Charging Session - Lock Failure**
Test case Id	TC_024_CS
Description	This scenario is used to report a connector lock failure.
Purpose	To test if the Charge Point is able to report a connector lock failure and does not start a transaction when it occurs.
Prerequisite(s)	The Charge Point does not have a fixed cable.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Start Charging Session - Lock Failure**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetConfiguration.conf**	[This step will be executing during the Before steps] **1.** The Central System sends a **GetConfiguration.req**
	4. The Charge Point responds with a **RemoteStartTransaction.conf**	[This step will be executing during the Before steps] **3.** The Central System sends a **RemoteStartTransaction.req**
	[If AuthorizeRemoteTxRequests = true (from step 2), send an Authorize.req.] [This step will be executing during the Before steps] **5.** The Charge Point sends an **Authorize.req**	**6.** The Central System responds with an **Authorize.conf**
	[This step will be executing during the Before steps] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
	[EV driver plugs in the cable halfway.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **GetConfiguration.conf**) The **configurationKey.key** is *AuthorizeRemoteTxRequests* * Step 4: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 7: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 9: (Message: **StatusNotification.req**) **errorCode** is *ConnectorLockFailure* **status** is *Faulted*	* Step 1: (Message: **GetConfiguration.req**) The **key** is *AuthorizeRemoteTxRequests* * Step 6: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	The Charging Station does NOT start a transaction.	n/a

2.11. Core Profile - Remote Actions Non-Happy Flow

2.11.1. Remote Start Charging Session – Rejected

Table 31. Test Case Id: TC_026_CS

Test case name	**Remote Start Charging Session – Rejected**	
Test case Id	TC_026_CS	
Description	This scenario is used to reject a RemoteStartTransaction.req, when a transaction is already ongoing on the requested connector.	
Purpose	To test if the Charge Point rejects a RemoteStartTransaction.req, when a transaction is already ongoing on the requested connector.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** - The value for "LocalPreAuthorize" is "false".	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **RemoteStartTransaction.conf**	**1.** The Central System sends a **RemoteStartTransaction.req**
Tool validation(s)	* Step 2: (Message: **RemoteStartTransaction.conf**) **status** is *Rejected*	* Step 1: (Message: **RemoteStartTransaction.req**) **connectorId** is the same connectorId used in ReusableState *Charging*
Expected result(s) / behaviour	n/a	n/a

2.11.2. Remote start transaction - connector id shall not be 0

Table 32. Test Case Id: TC_027_CS

Test case name	**Remote start transaction - connector id shall not be 0**	
Test case Id	TC_027_CS	
Description	This scenario is used to reject a RemoteStartTransaction.req on connectorId = 0.	
Purpose	To test if the Charge Point rejects a RemoteStartTransaction.req on connectorId = 0.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **RemoteStartTransaction.conf** OR with a CallError	**1.** The Central System sends a **RemoteStartTransaction.req**
Tool validation(s)	* Step 2: (Message: **RemoteStartTransaction.conf**) **status** is *Rejected*	* Step 1: (Message: **RemoteStartTransaction.req**) **connectorId** is *0*
Expected result(s) / behaviour	n/a	n/a

2.11.3. Remote Stop Transaction – Rejected

Table 33. Test Case Id: TC_028_CS

Test case name	**Remote Stop Transaction – Rejected**	
Test case Id	TC_028_CS	
Description	This scenario is used to reject a RemoteStopTransaction.req, when an unknown transactionId is given.	
Purpose	To test if the the Charge Point rejects a RemoteStopTransaction.req, when an unknown transactionId is given.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **RemoteStopTransaction.conf**	**1.** The Central System sends a **RemoteStopTransaction.req** with **transactionId** is *<Unknown transactionId>*
Tool validation(s)	* Step 2: (Message: **RemoteStopTransaction.conf**) **status** is *Rejected*	n/a
Expected result(s) / behaviour	n/a	n/a

2.12. Core Profile - Unlocking Non-happy flow

2.12.1. Unlock Connector – Unlock Failure

Table 34. Test Case Id: TC_030_CS

Test case name	**Unlock Connector – Unlock Failure**	
Test case Id	TC_030_CS	
Description	This scenario is used to report a connector lock failure.	
Purpose	To test if the Charge Point is able to report a connector lock failure.	
Prerequisite(s)	Ensure the Charge Point is in a state where a connector lock failure can be triggered.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *UnlockFailed*	n/a
Expected result(s) / behaviour	n/a	n/a

2.12.2. Unlock Connector – Unknown Connector

Table 35. Test Case Id: TC_031_CS

Test case name	**Unlock Connector – Unknown Connector**	
Test case Id	TC_031_CS	
Description	This scenario is used to reject an UnlockConnector.req, when an unknown connectorId is given.	
Purpose	To test if the Charge Point reacts correctly when receiving an UnlockConnector.req with an unknown connectorId.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req** with **connectorId** is *<Unknown connectorId>*
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *NotSupported*	n/a
Expected result(s) / behaviour	n/a	n/a

2.13. Core Profile - Power Failure Non-Happy Flow

2.13.1. Power failure boot charging point - configured to stop transaction(s) before going down

Table 36. Test Case Id: TC_032_1_CS

Test case name	**Power failure boot charging point - configured to stop transaction(s) before going down**	
Test case Id	TC_032_1_CS	
Description	This scenario is used to stop all transactions before going down, when a power failure occurs.	
Purpose	To test if the Charge Point first stops all transactions before going down, when a power failure occurs.	
Prerequisite(s)	The Charge Point has a back-up power source and thereby is configured to stop transactions before going down.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Disconnect the power of the Charge Point.] **1.** The Charge Point sends a **StopTransaction.req**	**2.** The Central System responds with a **StopTransaction.conf**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	[Reconnect the power of the Charge Point.] **5.** The Charge Point sends a **BootNotification.req**	**6.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId = 0.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StopTransaction.req**) **reason** is *PowerLoss* * Step 3: (Message: **StatusNotification.req**) **status** is *Finishing* * Step 7: (Message: **StatusNotification.req**) **connectorId** is *<The connector which had the ongoing transaction>* **status** is *Finishing OR Preparing* (Message: **StatusNotification.req**) *The other StatusNotification messages.* **status** is *Available*	* Step 6: (Message: **BootNotification.conf**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.13.2. Power failure boot charging point-configured to stop transaction(s)

Table 37. Test Case Id: TC_032_2_CS

Test case name	**Power failure boot charging point-configured to stop transaction(s)**
Test case Id	TC_032_2_CS
Description	This scenario is used to stop transaction all transactions, when a power failure occurred.
Purpose	To test if the Charge Point first stops all transactions after going down, when a power failure occurs.
Prerequisite(s)	The Charge Point does NOT have a back-up power source and thereby is configured to stop transactions after going down.

Test case name	**Power failure boot charging point-configured to stop transaction(s)**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Disconnect and reconnect the power of the Charge Point.] **1.** The Charge Point sends a **BootNotification.req**	**2.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId = 0.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	[When transaction is resumed by Charge Point (**status** is **Charging**) EV driver authorizes / swipes the card with the idTag which was used for the transaction to manually stop the transaction]	
	5. The Charge Point sends a **StopTransaction.req**	**6.** The Central System responds with a **StopTransaction.conf**
	[Only send when not already notified of the status Finishing.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Steps 3-7: The order in which the messages are sent may be different. * Step 3: (Message: **StatusNotification.req**) **connectorId** is *<The connector which had the ongoing transaction>* **status** is *Preparing*, *Finishing* OR *Charging* (intermediate status *unavailable* or *available* are allowed) (Message: **StatusNotification.req**) *The other StatusNotification messages.* **status** is *Available* * Step 5: (Message: **StopTransaction.req**) **reason** is *Local* or omitted when transaction was manually stopped **reason** is *PowerLoss* when transaction was stopped by charger due to power loss * Step 7: (Message: **StatusNotification.req**) **status** is *Preparing* or *Finishing*	* Step 2: (Message: **BootNotification.conf**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.13.3. Power Failure with Unavailable Status

Table 38. Test Case Id: TC_034_CS

Test case name	**Power Failure with Unavailable Status**
Test case Id	TC_034_CS
Description	This scenario is used to persist the status of the connectors, when a power failure occurs.
Purpose	To test if the Charge Point persists the status of the connectors, when a power failure occurs.
Prerequisite(s)	n/a

Test case name	**Power Failure with Unavailable Status**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeAvailability.conf**	**1.** The Central System sends a **ChangeAvailability.req**
	[Send per connector and connectorId = 0.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	[Disconnect and reconnect the power of the Charge Point.] **5.** The Charge Point sends a **BootNotification.req**	**6.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId = 0.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **ChangeAvailability.conf**) The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) **status** is *Unavailable* * Step 7: (Message: **StatusNotification.req**) **status** is *Unavailable*	* Step 1: (Message: **ChangeAvailability.req**) The **connectorId** is *0* The **type** is *Inoperative* * Step 6: (Message: **BootNotification.conf**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.14. Core Profile - Offline behavior Non-Happy Flow

2.14.1. Connection Loss During Transaction

Table 39. Test Case Id: TC_036_CS

Test case name	**Connection Loss During Transaction**	
Test case Id	TC_036_CS	
Description	This scenario is used to cache meter values, when a connection loss occurred during a transaction.	
Purpose	To test if the Charge Point is able to handle a connection loss during a transaction, without (for example) losing meter values.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** **MeterValueSampleInterval** is *<Configured MeterValueSampleInterval>*	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Remove the connectivity between the Charge Point and the Central System.] [Wait till charge point sends few meter values (at least 3)] [Restore the connectivity between the Charge Point and the Central System.] [Charge Point sends all queued meter values.] **1.** The Charge Point sends a **MeterValues.req**	**2.** The Central System sends a **MeterValues.conf**
Tool validation(s)	* Step 1: (Message: **MeterValues.req**) - All queued meter values need to be sent in chronological order (Also before sending any new meter values). - Between the reported timestamps need to be a number of seconds equal to the <configured MeterValueSampleInterval>	n/a
Expected result(s) / behaviour	n/a	n/a

2.14.2. Offline Start Transaction - Valid IdTag

Table 40. Test Case Id: TC_037_1_CS

Test case name	**Offline Start Transaction - Valid IdTag**
Test case Id	TC_037_1_CS
Description	This scenario is used to start a transaction, while being offline.
Purpose	To test if the Charge Point is able to start a transaction, while being offline and is able to queue transaction-related messages, after restoring the connection.
Prerequisite(s)	The Charge Point supports offline transactions using Local Authorization List, Authorization Cache or Unknown Offline Authorization.

Test case name	**Offline Start Transaction - Valid IdTag**	
Before	**Configuration State(s):** - *LocalAuthorizeOffline* is *true*. - *LocalAuthListEnabled* is *true*. (If implemented) - *AuthorizationCacheEnabled* is *true*. (If implemented) - *AllowOfflineTxForUnknownId* is *true*. (If implemented)	
	Memory State(s): - *IdTagLocalAuthList* for <Configured valid IdTag>. (If implemented) - *IdTagCached* for <Configured valid IdTag>. (If implemented)	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Remove connectivity between Charge Point and Central System.] [EV Driver starts offline a transaction with a valid idTag.] [Restore connectivity between Charge Point and Central System.] **1.** The Charge Point sends a **StartTransaction.req**	**2.** The Central System responds with a **StartTransaction.conf**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 2: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.14.3. Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = false

Table 41. Test Case Id: TC_037_2_CS

Test case name	**Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = false**
Test case Id	TC_037_2_CS
Description	This scenario is used to start a transaction, while being offline.
Purpose	To test if the Charge Point is able to start a transaction, while being offline and is able to queue transaction-related messages, after restoring the connection.
Prerequisite(s)	The Charge Point supports offline transactions using Local Authorization List, Authorization Cache or Unknown Offline Authorization.
Before	**Configuration State(s):** - *LocalAuthorizeOffline* is *true*. - *LocalAuthListEnabled* is *true*. (If implemented) - *AuthorizationCacheEnabled* is *true*. (If implemented) - *AllowOfflineTxForUnknownId* is *true*. (If implemented) - *StopTransactionOnInvalidId* is *false*. - *MaxEnergyOnInvalidId* is *0*. (If implemented)
	Memory State(s): - *IdTagLocalAuthList* for <Configured invalid IdTag>. (If implemented) - *IdTagCached* for <Configured invalid IdTag>. (If implemented)
	Reusable State(s): n/a

Test case name	**Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = false**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Remove connectivity between Charge Point and Central System.] [EV Driver starts offline a transaction with an invalid idTag.] [Restore connectivity between Charge Point and Central System.] **1.** The Charge Point sends a **StartTransaction.req**	**2.** The Central System responds with a **StartTransaction.conf**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) **status** is *Charging* * Step 5: (Message: **StatusNotification.req**) **status** is *SuspendedEVSE*	* Step 2: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Invalid*
Expected result(s) / behaviour	n/a	n/a

2.14.4. Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = true

Table 42. Test Case Id: TC_037_3_CS

Test case name	**Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = true**
Test case Id	TC_037_3_CS
Description	This scenario is used to start a transaction, while being offline.
Purpose	To test if the Charge Point is able to start a transaction, while being offline and is able to queue transaction-related messages, after restoring the connection.
Prerequisite(s)	The Charge Point supports offline transactions using Local Authorization List, Authorization Cache or Unknown Offline Authorization.
Before	**Configuration State(s):** - *LocalAuthorizeOffline* is *true*. - *LocalAuthListEnabled* is *true*. (If implemented) - *AuthorizationCacheEnabled* is *true*. (If implemented) - *AllowOfflineTxForUnknownId* is *true*. (If implemented) - *StopTransactionOnInvalidId* is *true*. - *MaxEnergyOnInvalidId* is *0*. (If implemented)
	Memory State(s): - *IdTagLocalAuthList* for <Configured invalid IdTag>. (If implemented) - *IdTagCached* for <Configured invalid IdTag>. (If implemented)
	Reusable State(s): n/a

Test case name	**Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = true**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Remove connectivity between Charge Point and Central System.] [EV Driver starts offline a transaction with an invalid idTag.] [Restore connectivity between Charge Point and Central System.] **1.** The Charge Point sends a **StartTransaction.req**	**2.** The Central System responds with a **StartTransaction.conf**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5. The Charge Point sends a **StopTransaction.req**	**6.** The Central System responds with a **StopTransaction.conf**
	7. The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) **status** is *Charging* * Step 5: (Message: **StopTransaction.req**) **reason** is *DeAuthorized* * Step 7 (Message: **StatusNotification.req**) **status** is *Finishing*	* Step 2: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Invalid*
Expected result(s) / behaviour	n/a	n/a

2.14.5. Offline Stop Transaction

Table 43. Test Case Id: TC_038_CS

Test case name	**Offline Stop Transaction**	
Test case Id	TC_038_CS	
Description	This scenario is used to stop a transaction, while the Charge Point is offline.	
Purpose	To test if the Charge Point is able to stop a transaction, while being offline.	
Prerequisite(s)	The Charge Point supports local stop transaction.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Remove the connectivity between the Charge Point and the Central System.] [The EV Driver stops the transaction, while still offline.] [Restore the connectivity between the Charge Point and the Central System.] [Steps 1 and 3 may be reversed] **1.** The Charge Point sends a **StopTransaction.req**	**2.** The Central System responds with a **StopTransaction.conf**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**

Test case name	**Offline Stop Transaction**	
Tool validation(s)	* Step 1: (Message: **StopTransaction.req**) **reason** is *Local* or is **omitted** * Step 3 (Message: **StatusNotification.req**) **status** is *Finishing*	n/a
Expected result(s) / behaviour	n/a	n/a

2.14.6. Offline Transaction

Table 44. Test Case Id: TC_039_CS

Test case name	**Offline Transaction**	
Test case Id	TC_039_CS	
Description	This scenario is used to start and stop a transaction, while the Charge Point is offline.	
Purpose	To test if the Charge Point is able to start and stop a transaction, while being offline and if it is able to queue all the transaction-related messages.	
Prerequisite(s)	The Charge Point supports offline transactions using Local Authorization List, Authorization Cache or Unknown Offline Authorization. The Charge Point supports local stop transaction.	
Before	**Configuration State(s):** - *LocalAuthorizeOffline* is *true*. - *LocalAuthListEnabled* is *true*. (If implemented) - *AuthorizationCacheEnabled* is *true*. (If implemented) - *AllowOfflineTxForUnknownId* is *true*. (If implemented)	
	Memory State(s): - *IdTagLocalAuthList* for <Configured valid IdTag>. (If implemented) - *IdTagCached* for <Configured valid IdTag>. (If implemented)	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Remove connectivity between Charge Point and Central System.] [EV Driver starts offline a transaction.] [EV Driver stops offline a transaction.] [EV driver unplugs the cable.] [Restore connectivity between Charge Point and Central System.] **1.** The Charge Point sends a **StartTransaction.req**	**2.** The Central System responds with a **StartTransaction.conf**
	3. The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
Tool validation(s)	* Step 3: (Message: **StopTransaction.req**) **reason** is *Local* or is **omitted**	* Step 2: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.15. Core Profile - Configuration Keys Non-Happy Flow

2.15.1. Configuration key - NotSupported

Table 45. Test Case Id: TC_040_1_CS

Test case name	**Configuration key - NotSupported**	
Test case Id	TC_040_1_CS	
Description	This scenario is used to reject an unknown configuration key.	
Purpose	To test if the Charge Point is able to notify the Central System that it does not support the given configuration key.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) The **status** is *NotSupported*	* Step 1: (Message: **ChangeConfiguration.req**) The **key** is *Testing* **value** is *true*
Expected result(s) / behaviour	n/a	n/a

2.15.2. Configuratoin key - Invalid value

Table 46. Test Case Id: TC_040_2_CS

Test case name	**Configuratoin key - Invalid value**	
Test case Id	TC_040_2_CS	
Description	This scenario is used to reject setting a configuration key, when an incorrect value is given.	
Purpose	To test if the Charge Point is able to reject setting a configuration key, when an incorrect value is given.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeConfiguration.conf** OR with a CallError.	**1.** The Central System sends a **ChangeConfiguration.req**
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) The **status** is *Rejected* OR (Message: **CallError** **ErrorCode** is *PropertyConstraintViolation*.	* Step 1: (Message: **ChangeConfiguration.req**) The **key** is *MeterValueSampleInterval* **value** is *-1*

Test case name	**Configuratoin key - Invalid value**	
Expected result(s) / behaviour	n/a	n/a

2.16. Core Profile - Fault Behavior Non-Happy Flow

2.16.1. Fault Behavior

Table 47. Test Case Id: TC_041_CS

Test case name	**Fault Behavior**	
Test case Id	TC_041_CS	
Description	This scenario is used to refuse starting a transaction, when the Charge Point in fault state.	
Purpose	To test if the Charge Point refuses starting a transaction, when it is in fault state.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[Set the Charge Point in fault state.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	3. [The EV Driver tries to start a transaction.] [The Charge Point does not start a transaction.]	
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Faulted* * Step 3: The tool waits for *<Configured Transaction Duration>* to verify that no transaction is started.	
Expected result(s) / behaviour	n/a	n/a

2.17. Local Authorization List

2.17.1. Get Local List Version

Get Local List Version (not supported)

Table 48. Test Case Id: TC_042_1_CS

Test case name	**Get Local List Version (not supported)**
Test case Id	TC_042_1_CS
Description	The Central System can request a Charge Point for the version number of the Local Authorization List.
Purpose	Check whether the Charge Point is able to provide the local list version, when requested.
Prerequisite(s)	The Charge Point does not support the Local Auth List Management feature profile or allows localAuthListEnabled=false.
Before	**Configuration State(s):** - *LocalAuthListEnabled* is *false*. (If implemented)
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	Get Local List Version (not supported)	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetLocalListVersion.conf**. OR with a CallError.	**1.** The Central System sends a **GetLocalListVersion.req**.
Tool validation(s)	* Step 2: (Message: **GetLocalListVersion.conf**) **listVersion** is *-1* OR (Message: **CallError** **ErrorCode** is *NotSupported*.	n/a
Expected result(s) / behaviour	n/a	n/a

Get Local List Version (empty)

Table 49. Test Case Id: TC_042_2_CS

Test case name	Get Local List Version (empty)	
Test case Id	TC_042_2_CS	
Description	The Central System can request a Charge Point for the version number of the Local Authorization List.	
Purpose	Check whether the Charge Point is able to provide the local list version as 0, when the list is empty.	
Prerequisite(s)	The Charge Point does support the Local Auth List Management feature profile.	
Before	**Configuration State(s):** - *LocalAuthListEnabled* is *true*.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SendLocalList.conf**.	**1.** The Central System sends a **SendLocalList.req**.
	4. The Charge Point responds with a **GetLocalListVersion.conf**.	**3.** The Central System sends a **GetLocalListVersion.req**.
Tool validation(s)	* Step 2: (Message: **SendLocalList.conf**) **status** is *Accepted* * Step 4: (Message: **GetLocalListVersion.conf**) **listVersion** is *0*	* Step 1: (Message: **SendLocalList.req**) **listVersion** is *1* **localAuthorizationList** is *omitted* **updateType** is *Full*
Expected result(s) / behaviour	n/a	n/a

2.17.2. Send Local Authorization List

Send Local Authorization List

Table 50. Test Case Id: TC_043_CS

Test case name	Send Local Authorization List
Test case Id	TC_043_CS
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.
Purpose	Check whether a Local Authorization List can be sent to a Charge Point to authorize an EV driver
Prerequisite(s)	The Charge Point supports the Local Auth List Management feature profile.

Test case name	Send Local Authorization List	
Before	**Configuration State(s):** - **LocalAuthListEnabled** is *true*.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SendLocalList.conf**	**1.** The Central System sends a **SendLocalList.req**
	4. The Charge Point responds with a **SendLocalList.conf**	**3.** The Central System sends a **SendLocalList.req**
Tool validation(s)	* Step 2: (Message: **SendLocalList.conf**) - **Status** should be *Accepted* * Step 4: (Message: **SendLocalList.conf**) - **Status** should be *Accepted*	* Step 1: (Message: **SendLocalList.req**) - **updateType** is *Full* * Step 3: (Message: **SendLocalList.req**) - **UpdateType** is *Differential*
Expected result(s) / behaviour	The Charge Point can Authorize EV drivers that have an IdToken that is on the local authorization list.	n/a

Send Local Authorization List - NotSupported

Table 51. Test Case Id: TC_043_1_CS

Test case name	Send Local Authorization List - NotSupported	
Test case Id	TC_043_1_CS	
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.	
Purpose	Check whether a Charge Point can refuse a sent Local Authorization List if it does not support it.	
Prerequisite(s)	The Charge Point does not support the Local Auth List Management feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SendLocalList.conf** to the Central System. OR with a CallError.	**1.** The Central System sends a **SendLocalList.req** to the Charge Point.
Tool validation(s)	* Step 2: (Message: **SendLocalList**) - **Status** should be *NotSupported* OR (Message: **CallError** **ErrorCode** is *NotSupported*.	* Step 1: (Message: **SendLocalList.req**) - **updateType** should be *Full*
Expected result(s) / behaviour	The Charge Point cannot locally authorize EV drivers that have an IdToken that is on the local authorization list that was sent.	n/a

Send Local Authorization List - VersionMismatch

Table 52. Test Case Id: TC_043_2_CS

Test case name	**Send Local Authorization List - VersionMismatch**	
Test case Id	TC_043_2_CS	
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.	
Purpose	Check whether a Charge Point can refuse a sent Local Authorization List.	
Prerequisite(s)	The Charge Point supports the Local Auth List Management feature profile.	
Before	**Configuration State(s):** - **LocalAuthListEnabled** is *true*.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SendLocalList.conf**	**1.** The Central System sends a **SendLocalList.req**
	4. The Charge Point responds with a **GetLocalListVersion.conf**	**3.** The Central System sends a **GetLocalListVersion.req**
	6. The Charge Point responds with a **SendLocalList.conf**	**5.** The Central System sends a **SendLocalList.req**
	8. The Charge Point responds with a **GetLocalListVersion.conf**	**7.** The Central System sends a **GetLocalListVersion.req**
	10. The Charge Point responds with a **SendLocalList.conf**	**9.** The Central System sends a **SendLocalList.req**
	12. The Charge Point responds with a **GetLocalListVersion.conf**	**11.** The Central System sends a **GetLocalListVersion.req**
Tool validation(s)	* Step 2: (Message: **SendLocalList.conf**) - **Status** should be *Accepted* * Step 4: (Message: **GetLocalListVersion.conf**) - **listVersion** should be *2* * Step 6: (Message: **SendLocalList.conf**) - **Status** should be *Accepted* * Step 8: (Message: **GetLocalListVersion.conf**) - **listVersion** should be *5* * Step 10: (Message: **SendLocalList.conf**) - **Status** should be *VersionMismatch* * Step 12: (Message: **GetLocalListVersion.conf**) - **listVersion** should be *5*	* Step 1: (Message: **SendLocalList.req**) - **updateType** is *Full* - **listVersion** is *2* * Step 5: (Message: **SendLocalList.req**) - **updateType** is *Differential* - **listVersion** is *5* * Step 9: (Message: **SendLocalList.req**) - **updateType** is *Differential* - **listVersion** is *4*
Expected result(s) / behaviour	The Charge Point rejects a LocalList with an old version number.	n/a

Send Local Authorization List - Failed

Table 53. Test Case Id: TC_043_3_CS

Test case name	**Send Local Authorization List - Failed**
Test case Id	TC_043_3_CS
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.
Purpose	Check whether a Charge Point can refuse a sent Local Authorization List.

Test case name	Send Local Authorization List - Failed	
Prerequisite(s)	- The Charge Point is in a state in which it will fail to set a Local List from the Central System. - The Charge Point supports the Local Auth List Management feature profile.	n/a
Before	**Configuration State(s):** - **LocalAuthListEnabled** is *true*.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SendLocalList.conf** to the Central System.	**1.** The Central System sends a **SendLocalList.req** to the Charge Point.
Tool validation(s)	* Step 2: (Message: **SendLocalList**) - **Status** should be *Failed*	* Step 1: (Message: **SendLocalList.req**) - **updateType** should be *Full* - **listVersion** should be *2*
Expected result(s) / behaviour	n/a	n/a

2.17.3. Regular Start Charging Session – Id in Local Authorization List

Table 54. Test Case Id: TC_008_CS

Test case name	Regular Start Charging Session – Id in Local Authorization List	
Test case Id	TC_008_CS	
Description	This scenario is used to authorize a transaction using the Local Authorization List.	
Purpose	To test if the Charge Point can start a transaction using the Local Authorization List.	
Prerequisite(s)	Local Auth List Management feature profile is supported AND The configuration key *AuthorizeRemoteTxRequests* does not have an accessibility of *ReadOnly* in combination with the value *false*.	
Before	**Configuration State(s):** - *LocalPreAuthorize* is *true*. - *AuthorizationCacheEnabled* is *false*. (If implemented) - *LocalAuthListEnabled* is *true*. - *AuthorizeRemoteTxRequests* is *true*.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetLocalListVersion.conf**.	**1.** The Central System sends a **GetLocalListVersion.req**.
	4. The Charge Point responds with a **SendLocalList.conf**.	**3.** The Central System sends a **SendLocalList.req**.
	[EV driver plugs in the cable.] **5.** The Charge Point sends a **StatusNotification.req**.	**6.** The Central System responds with a **StatusNotification.conf**.
	8. The Charge Point sends a **RemoteStartTransaction.conf**.	**7.** The Central System sends a **RemoteStartTransaction.req**.
	9. The Charge Point sends a **StartTransaction.req**.	**10.** The Central System responds with a **StartTransaction.conf**.
	11. The Charge Point sends a **StatusNotification.req**.	**12.** The Central System responds with a **StatusNotification.conf**.

Test case name	**Regular Start Charging Session – Id in Local Authorization List**	
Tool validation(s)	* Step 4: (Message: **SendLocalList.conf**) **status** is *Accepted* * Step 5: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 8: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 11: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 3: (Message: **SendLocalList.req**) **updateType** is *Full* **localAuthorizationList[0].idTag** is *<Configured valid IdTag>* **localAuthorizationList[0].idTagInfo.status** is *Accepted* * Step 10: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

2.18. FirmwareManagement

2.18.1. Firmware Update - Download and Install

Table 55. Test Case Id: TC_044_1_CS

Test case name	**Firmware Update - Download and Install**
Test case Id	TC_044_1_CS
Description	The firmware of a Charge Point is updated.
Purpose	Check whether the Charge Point can update its firmware.
Prerequisite(s)	- The Charge Point supports the Firmware Management feature profile and a dummy firmware is prepared. - Based on the configuration key SupportedFileTransferProtocols. FTP, FTPS, HTTP, HTTPS. The tester has to setup a server which supports one of the specified protocols. - A valid firmware needs to stored at the server and configured at the Firmware Download URL.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Firmware Update - Download and Install**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UpdateFirmware.conf**	**1.** The Central System sends a **UpdateFirmware.req**
	[Before downloading the firmware the Charge Point MAY set all connectors to Unavailable. If the Charge Point supports installation of firmware during a charging session, the Charge Point MAY install the firmware after only setting all other connectors to Unavailable.] [The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **FirmwareStatusNotification.req**	**4.** The Central System responds with a **FirmwareStatusNotification.conf**
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **FirmwareStatusNotification.req**	**6.** The Central System responds with a **FirmwareStatusNotification.conf**
	[The Charge Point starts installing the firmware] **7.** The Charge Point sends a **FirmwareStatusNotification.req**	**8.** The Central System responds with a **FirmwareStatusNotification.conf**
	9. The Charge Point sends a **BootNotification.req**	**10.** The Central System responds with a **BootNotification.conf**
	11. The Charge Point sends a **StatusNotification.req**	**12.** The Central System responds with a **StatusNotification.conf**
	13. The Charge Point sends a **FirmwareStatusNotification.req**	**14.** The Central System responds with a **FirmwareStatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (Message: **FirmwareStatusNotification.req**) The **status** is *Installing* * Step 9 / 13: The messages can be in a different order, but the described order is recommended. * Step 11: (Message: **StatusNotification.req**) The **status** is *Available* * Step 13: (Message: **FirmwareStatusNotification.req**) The **status** is *Installed*	* Step 1: (Message: **UpdateFirmware.req**) The **firmware.location** is *<Firmware Download URL from test data>*
Expected result(s) / behaviour	The Charge Point handles the firmware update correctly and is Available after the update.	n/a

2.18.2. Firmware Update - Download Failed

Table 56. Test Case Id: TC_044_2_CS

Test case name	**Firmware Update - Download Failed**
Test case Id	TC_044_2_CS
Description	The firmware of a Charge Point is being updated, but downloading the firmware fails.
Purpose	Check whether the Charge Point can exchange valid messages for a firmware update in case downloading of the firmware fails.
Prerequisite(s)	The Charge Point supports the Firmware Management feature profile.

Test case name	Firmware Update - Download Failed	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UpdateFirmware.conf**	**1.** The Central System sends a **UpdateFirmware.req**
	[Before downloading the firmware the Charge Point MAY set all connectors to Unavailable.] [The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **FirmwareStatusNotification.req**	**4.** The Central responds with a **FirmwareStatusNotification.conf**
	[Downloading the firmware fails] **5.** The Charge Point sends a **FirmwareStatusNotification.req**	**6.** The Central responds with a **FirmwareStatusNotification.conf**
Tool validation(s)	* Step 3: (This message is optional, because the download may fail immediately) (Message: **FirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **FirmwareStatusNotification.req**) The **status** is *DownloadFailed*	* Step 1: (Message: **UpdateFirmware.req**) **location** is *ftp://127.0.0.1:21/download_fail.fwi* **retries** is *0*
Expected result(s) / behaviour	Old firmware remains active, Charge Point becomes *Available* again after being set to *Unavailable* when downloading the firmware.	n/a

2.18.3. Firmware Update - Installation Failed

Table 57. Test Case Id: TC_044_3_CS

Test case name	Firmware Update - Installation Failed
Test case Id	TC_044_3_CS
Description	The firmware of a Charge Point is being updated, but the installation fails.
Purpose	Check whether the Charge Point can exchange valid messages to update the firmware of a Charge Point in case the installation fails.
Prerequisite(s)	- Based on the configuration key SupportedFileTransferProtocols. FTP, FTPS, HTTP, HTTPS. The tester has to setup a server which supports one of the specified protocols. - An invalid firmware needs to stored at the server and configured at the Firmware Download URL.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Firmware Update - Installation Failed**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **UpdateFirmware.conf**	**1.** The Central System sends a **UpdateFirmware.req**
	[Before downloading the firmware the Charge Point MAY set all connectors to Unavailable. If the Charge Point supports installation of firmware during a charging session, the Charge Point MAY install the firmware after only setting all other connectors to Unavailable.] [The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **FirmwareStatusNotification.req**	**4.** The Central responds with a **FirmwareStatusNotification.conf**
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **FirmwareStatusNotification.req**	**6.** The Central responds with a **FirmwareStatusNotification.conf**
	[The Charge Point starts installing the firmware] **7.** The Charge Point sends a **FirmwareStatusNotification.req**	**8.** The Central responds with a **FirmwareStatusNotification.conf**
	[This step is optional.] **9.** The Charge point reboots and sends a **BootNotification.req**	**10.** The Central System responds with a **BootNotification.conf**
	11. The Charge Point sends a **FirmwareStatusNotification.req**	**12.** The Central responds with a **FirmwareStatusNotification.conf**
	[This step is optional. The Charge Point reports the status of all connectors after a boot.] **13.** The Charge Point sends a **StatusNotification.req**	**14.** The Central responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (This message is optional, because the installation may fail immediately) (Message: **FirmwareStatusNotification.req**) The **status** is *Installing* * Step 9 / 11 / 13: The messages can be in a different order. * Step 11: (Message: **FirmwareStatusNotification.req**) The **status** is *InstallationFailed* * Step 13: (Message: **StatusNotification.req**) The **status** is *Available*	* Step 1: (Message: **UpdateFirmware.req**) **location** is *<The location of a NOT supported file>*
Expected result(s) / behaviour	n/a	n/a

2.19. Diagnostics

2.19.1. Get Diagnostics

Table 58. Test Case Id: TC_045_1_CS

Test case name	**Get Diagnostics**	
Test case Id	TC_045_1_CS	
Description	The Charge Point uploads a diagnostics log to a specified location based on a request of the Central System.	
Purpose	The purpose of this test case it to check whether the Charge Point can upload its diagnostics.	
Prerequisite(s)	Based on the configuration key SupportedFileTransferProtocols. FTP, FTPS, HTTP, HTTPS. The tester has to set up a server which supports one of the specified protocols.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetDiagnostics.conf** to the Central System.	**1.** The Central System sends a **GetDiagnostics.req** to the Charge Point.
	[The Charge Point starts uploading the diagnostics log.] **3.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**4.** The Central System responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.
	[The Charge Point has finished uploading the diagnostics log.] **5.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**6.** The Central responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.
Tool validation(s)	* Step 3: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Uploading* * Step 5: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Uploaded*	* Step 1: (Message: **GetDiagnostics.req**) The **location** is *<Configured log location>*
Expected result(s) / behaviour	The Charge Point has uploaded the diagnostics log to the **location** that was sent in step 1.	n/a

2.19.2. Get Diagnostics - Upload Failed

Table 59. Test Case Id: TC_045_2_CS

Test case name	**Get Diagnostics - Upload Failed**
Test case Id	TC_045_2_CS
Description	When getting the diagnostics of a Charge Point, the upload of the log fails.
Purpose	Check whether the Charge Point can exchange valid messages for the situation that the upload fails when getting the diagnostics.
Prerequisite(s)	n/a
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Get Diagnostics - Upload Failed**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetDiagnostics.conf** to the Central System.	**1.** The Central System sends a **GetDiagnostics.req** to the Charge Point.
	[The Charge Point starts uploading the diagnostics log.] **3.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**4.** The Central responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.
	[The Charge Point has failed uploading the diagnostics log.] **5.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**6.** The Central responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.
Tool validation(s)	* Step 3: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Uploading* * Step 5: (Message: **DiagnosticsStatusNotification.req**) The **status** is *UploadFailed*	* Step 1: (Message: **GetDiagnostics.req**) **retries** is *0* **location** is *ftp://127.0.0.1:21/files/failedLocation*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

2.20. Reservation

2.20.1. Reservation of a Connector

Reservation of a Connector - Local start transaction

Table 60. Test Case Id: TC_046_1_CS

Test case name	**Reservation of a Connector - Local start transaction**
Test case Id	TC_046_1_CS
Description	A Connector is reserved and a charging transaction takes place.
Purpose	Check whether the Charge Point can reserve a Connector.
Prerequisite(s)	The Charge Point supports the Reservation feature profile.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): - *SetConnectorUnavailable* for all unused connectors

Test case name	**Reservation of a Connector - Local start transaction**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
	3 The Charge Point sends a **StatusNotification.req** to the Central System	**4.** The Central System responds with a **StatusNotification.conf** to the Charge Point
	[EV driver authorizes / swipes a card (not the idTag from step 1)] [EV driver authorizes / swipes the card with the idTag from step 1] **5.** The Charge Point sends an optional **Authorize.req** to the Central System	**6.** The Central System responds with an **Authorize.conf** to the Charge Point
	7. The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
	[EV driver plugs in cable at the reserved Connector] **9.** The Charge Point sends a **StartTransaction.req**	**10.** The Central System responds with a **StartTransaction.conf**
	11 The Charge Point sends a **StatusNotification.req**	**12.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) - The **status** is *Reserved* * Step 5: (Message: **Authorize.req**) - The **idTag** matches the **idTag** from step 1. * Step 7: (Message: **StatusNotification.req**) - The **status** is *Preparing* * Step 9: (Message: **StartTransaction.req**) - The **reservationId** matches the **reservationId** from step 1. - The **idTag** matches the **idTag** from step 1. - The **idTag** and **reservationId** are included in the message. * Step 11: (Message: **StatusNotification.req**) - The **status** is *Charging*	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** is *<Configured ConnectorId>* - The **idTag** is *<Configured Valid IdTag>* * Step 8: (Message: **Authorize.conf**) - The **idTagInfo.status** is *Accepted* * Step 10: (Message: **StartTransaction.conf**) - The **status** is *Accepted*
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, only the **idTag** from the reservation can charge on the reserved Connector.	n/a

Reservation of a Connector - Remote start transaction

Table 61. Test Case Id: TC_046_2_CS

Test case name	**Reservation of a Connector - Remote start transaction**
Test case Id	TC_046_2_CS
Description	A Connector is reserved and a charging transaction takes place.
Purpose	Check whether the Charge Point can reserve a Connector.
Prerequisite(s)	The Charge Point supports the Reservation feature profile.

Test case name	**Reservation of a Connector - Remote start transaction**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5. *Charging* (Sending an **Authorize.req** is optional)	
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) - The **status** is *Reserved* * Step 5: - The **idTag** and **reservationId** from the **StartTransaction.req** matches the **idTag** and **reservationId** from step 1.	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** is *<Configured ConnectorId>* - The **idTag** is *<Configured Valid IdTag>*
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, only the **idTag** from the reservation can charge on the reserved Connector.	n/a

Reservation of a Connector - Expire

Table 62. Test Case Id: TC_047_CS

Test case name	**Reservation of a Connector - Expire**	
Test case Id	TC_047_CS	
Description	A Connector is reserved, a charging transaction could take place, but the reservation is not used (in time)	
Purpose	Check whether the Charge Point can exchange valid messages when the reservation is not used (in time).	
Prerequisite(s)	The Charge Point supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetConnectorUnavailable* for all unused connectors	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	[EV driver does not arrive at the reserved Connector before the expiry date] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[The tool will start a transaction with another valid id tag to ensure the reservation is expired.] **7.** *Charging*	

Test case name	**Reservation of a Connector - Expire**	
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** should be *Accepted* * Step 3: (Message: **StatusNotification.req**) - The **status** should be *Reserved* - The **connectorId** matches the **connectorId** from step 1 * Step 5: (Message: **StatusNotification.req**) - The **status** should be *Available* - The **connectorId** matches the **connectorId** from step 1	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** is *<Configured ConnectorId>* - The **expiryDate** is the current time plus <Configured Reservation Expiry Date Offset>
Expected result(s) / behaviour	After the expiry date, the Charge Point makes the *Reserved* connector *Available* again.	n/a

Reservation of a Connector - Faulted

Table 63. Test Case Id: TC_048_1_CS

Test case name	**Reservation of a Connector - Faulted**	
Test case Id	TC_048_1_CS	
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Faulted* is returned by the Charge Point.	
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.	
Prerequisite(s)	The Charge Point supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetConnectorFaulted*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Faulted*	* Step 1: (Message: **ReserveNow.req**) - **connectorId** is *<Configured ConnectorId>*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

Reservation of a Connector - Occupied

Table 64. Test Case Id: TC_048_2_CS

Test case name	**Reservation of a Connector - Occupied**
Test case Id	TC_048_2_CS
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Occupied* is returned by the Charge Point.
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.
Prerequisite(s)	The Charge Point supports the Reservation feature profile.

Test case name	**Reservation of a Connector - Occupied**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetConnectorOccupied*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Occupied*	* Step 1: (Message: **ReserveNow.req**) - **connectorId** is *<Configured ConnectorId>*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

Reservation of a Connector - Unavailable

Table 65. Test Case Id: TC_048_3_CS

Test case name	**Reservation of a Connector - Unavailable**	
Test case Id	TC_048_3_CS	
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Unavailable* is returned by the Charge Point.	
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.	
Prerequisite(s)	The Charge Point supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetConnectorUnavailable*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Unavailable*	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** is *<Configured ConnectorId>*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

Reservation of a Connector - Rejected

Table 66. Test Case Id: TC_048_4_CS

Test case name	**Reservation of a Connector - Rejected**
Test case Id	TC_048_4_CS
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Rejected* is returned by the Charge Point.
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.
Prerequisite(s)	The Charge Point does NOT support the Reservation feature profile.

Test case name	**Reservation of a Connector - Rejected**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Rejected*	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** is *<Configured ConnectorId>*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

2.20.2. Reservation of a Charge Point

Reservation of a Charge Point - Transaction

Table 67. Test Case Id: TC_049_CS

Test case name	**Reservation of a Charge Point - Transaction**	
Test case Id	TC_049_CS	
Description	A Charge Point / unspecified Connector is reserved and a charging transaction takes place.	
Purpose	Check whether the Charge Point can reserve an unspecified Connector.	
Prerequisite(s)	- The Charge Point supports the Reservation feature profile. - The value for **ReserveConnectorZeroSupported** is set to *true*.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetConnectorUnavailable* for all unused connectors	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	[The tool will start a transaction on the reserved connector] **5.** *Charging* (Sending an **Authorize.req** is optional)	
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** should be *Accepted* * Step 3: (Message: **StatusNotification.req**) - The **status** should be *Reserved* * Step 5: - The **idTag** and **reservationId** from the **StartTransaction.req** matches the **idTag** and **reservationId** from step 1.	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** is *0*
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, only the **idTag** from the reservation can charge, on any available connector of the Charge Point.	n/a

Reservation of a Charge Point - Faulted

Table 68. Test Case Id: TC_050_1_CS

Test case name	**Reservation of a Charge Point - Faulted**	
Test case Id	TC_050_1_CS	
Description	The Central System attempts to reserve a Charge Point, but the reservation is not made, instead the status *Faulted* is returned.	
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.	
Prerequisite(s)	- The Charge Point supports the Reservation feature profile. - **ReserveConnectorZeroSupported** is *true*	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetChargePointFaulted*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Faulted*	* Step 1: (Message: **ReserveNow.req**) - **connectorId** is *0*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

Reservation of a Charge Point - Occupied

Table 69. Test Case Id: TC_050_2_CS

Test case name	**Reservation of a Charge Point - Occupied**	
Test case Id	TC_050_2_CS	
Description	The Central System attempts to reserve a Charge Point, but the reservation is not made, instead the status *Occupied* is returned.	
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.	
Prerequisite(s)	- The Charge Point supports the Reservation feature profile. - **ReserveConnectorZeroSupported** is *true*	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetConnectorOccupied* for all connectors	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Occupied*	* Step 1: (Message: **ReserveNow.req**) - **connectorId** is *0*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

Reservation of a Charge Point - Unavailable

Table 70. Test Case Id: TC_050_3_CS

Test case name	**Reservation of a Charge Point - Unavailable**	
Test case Id	TC_050_3_CS	
Description	The Central System attempts to reserve a Charge Point, but the reservation is not made, instead the status *Unavailable* is returned.	
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.	
Prerequisite(s)	- The Charge Point supports the Reservation feature profile. - **ReserveConnectorZeroSupported** is *true*	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetChargePointUnavailable*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Unavailable*	* Step 1: (Message: **ReserveNow.req**) - **connectorId** is *0*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

Reservation of a Charge Point - Rejected

Table 71. Test Case Id: TC_050_4_CS

Test case name	**Reservation of a Charge Point - Rejected**	
Test case Id	TC_050_4_CS	
Description	The Central System attempts to reserve a Charge Point, but the reservation is not made, instead the status *Rejected* is returned.	
Purpose	Check whether the Charge Point is able to exchange messages in case that a reservation cannot be made.	
Prerequisite(s)	The Charge Point does NOT support the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point sends a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Rejected*	* Step 1: (Message: **ReserveNow.req**) - **connectorId** is *0*
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

2.20.3. Cancel Reservation

Cancel Reservation

Table 72. Test Case Id: TC_051_CS

Test case name	**Cancel Reservation**	
Test case Id	TC_051_CS	
Description	The Central System cancels an existing, not expired reservation.	
Purpose	Check whether the Charge Point is able to cancel a reservation.	
Prerequisite(s)	The Charge Point supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Reserved* with <Configured Valid IdTag>	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds a **CancelReservation.conf**	**1.** The Central System sends a **CancelReservation.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds a **StatusNotification.conf**
	5. *Charging* with <Configured Valid IdTag 2>	
Tool validation(s)	* Step 2: (Message: **CancelReservation.conf**) - **status** should be *Accepted* * Step 3: (Message: **StatusNotification.req**) - **status** should be *Available* - **connectorId** should match the connectorId used for reservation * Step 5: (Reusable state: **Charging**) - **reservationId** should be omitted.	* Step 1: (Message: **CancelReservation.req**) - **reservationId** matches the **reservationId** from the reusable state *Reserved*
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, cancelling only the reservation with the right reservationId.	n/a

Cancel Reservation - Rejected

Table 73. Test Case Id: TC_052_CS

Test case name	**Cancel Reservation - Rejected**	
Test case Id	TC_052_CS	
Description	The Central System tries to cancel reservation, but this request is rejected by the Charge Point.	
Purpose	Check whether the Charge Point is able to exchange messages in case of cancelling a reservation.	
Prerequisite(s)	The Charge Point supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Reserved*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **CancelReservation.conf**	**1.** The Central System sends a **CancelReservation.req**

Test case name	**Cancel Reservation - Rejected**	
Tool validation(s)	* Step 2: (Message: **CancelReservation.conf**) - **status** is *Rejected*	* Step 1: (Message: **CancelReservation.req**) - **reservationId** does NOT match the **reservationId** from reusable state *Reserved*
Expected result(s) / behaviour	The Charge Point rejects the unknown *reservationId* and does not cancel any reservation.	n/a

2.20.4. Use a reserved Connector with parentIdTag

Table 74. Test Case Id: TC_053_CS

Test case name	**Use a reserved Connector with parentIdTag**	
Test case Id	TC_053_CS	
Description	The Charge Point has been reserved and is used with a *parentIdTag*	
Purpose	Check whether the Charge Point is able to exchange messages for a reservation that is used by a *parentIdTag*	
Prerequisite(s)	- The Charge Point supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *SetConnectorUnavailable* for all unused connectors	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
	3 The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5. Execute Reusable State *Authorized* with <Configured Valid IdTag 2>	
	6. Manual Action: *EV driver plugs in the cable.*	
	7. The Charge Point sends a **StartTransaction.req**	**8.** The Central System responds with a **StartTransaction.conf**
	9. The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
	Note: Step 7 and step 9 may be reversed.	
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Accepted* * Step 3: (Message: **StatusNotification.req**) - **status** should be *Reserved* * Step 5: (Message: **StartTransaction.req**) - **reservationId** should match the **reservationId** from step 1.	* Step 1: (Message: **ReserveNow.req**) - **idTag** is *<Configured Valid IdTag 1>* - **parentIdTag** is *<Configured ParentId>*
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, the **parentIdTag** from the reservation can charge on the reserved Connector.	n/a

2.21. RemoteTrigger

2.21.1. Trigger Message

Table 75. Test Case Id: TC_054_CS

Test case name	**Trigger Message**	
Test case Id	TC_054_CS	
Description	The Central System triggers a message from the Charge Point	
Purpose	whether the Charge Point is able to provide the triggered message.	
Prerequisite(s)	The Charge Point supports the Remote Trigger feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **TriggerMessage.conf**	**1.** The Central System sends a **TriggerMessage.req**
	3. The Charge Point sends a **MeterValues.req**	**4.** The Central System responds with a **MeterValues.conf**
	6. The Charge Point responds with a **TriggerMessage.conf**	**5.** The Central System sends a **TriggerMessage.req**
	7. The Charge Point sends a **Heartbeat.req**	**8.** The Central System responds with a **Heartbeat.conf**
	10. The Charge Point responds with a **TriggerMessage.conf**	**9.** The Central System sends a **TriggerMessage.req**
	11. The Charge Point sends a **StatusNotification.req**	**12.** The Central System responds with a **StatusNotification.conf**
	14. The Charge Point responds with a **TriggerMessage.conf**	**13.** The Central System sends a **TriggerMessage.req**
	15. The Charge Point sends a **DiagnosticsStatusNotification.req**	**16.** The Central System responds with a **DiagnosticsStatusNotification.conf**
	18. The Charge Point responds with a **TriggerMessage.conf**	**17.** The Central System sends a **TriggerMessage.req**
	[The following message will be sent if implemented.] **19.** The Charge Point sends a **FirmwareStatusNotification.req**	**20.** The Central System responds with a **FirmwareStatusNotification.conf**
Tool validation(s)	* Step 2/6/10/14: (Message: **TriggerMessage.conf**) The **status** is *Accepted* * Step 15: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Idle* * Step 18: (Message: **TriggerMessage.conf**) The **status** is *Accepted* OR *NotImplemented* * Step 19: (Message: **FirmwareStatusNotification.req**) The **status** is *Idle*	* Step 1: (Message: **TriggerMessage.req**) **requestedMessage** should be *MeterValues* **connectorId** should be *<Configured ConnectorId>* * Step 5: (Message: **TriggerMessage.req**) **requestedMessage** should be *Heartbeat* * Step 9: (Message: **TriggerMessage.req**) **requestedMessage** should be *StatusNotification* **connectorId** should be *<Configured ConnectorId>* * Step 13: (Message: **TriggerMessage.req**) **requestedMessage** should be *DiagnosticsStatusNotification* * Step 17: (Message: **TriggerMessage.req**) **requestedMessage** should be *FirmwareStatusNotification*

Test case name	**Trigger Message**	
Expected result(s) / behaviour	n/a	n/a

2.21.2. Trigger Message - Rejected

Table 76. Test Case Id: TC_055_CS

Test case name	**Trigger Message - Rejected**	
Test case Id	TC_055_CS	
Description	The Central System triggers a message from the Charge Point, but the Charge Point rejects the message.	
Purpose	Check whether the Charge Point is able to reject a message triggered by the Central System.	
Prerequisite(s)	The Charge Point supports the Remote Trigger feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **TriggerMessage.conf** OR with a CallError	**1.** The Central System sends a **TriggerMessage.req**
Tool validation(s)	* Step 2: (Message: **TriggerMessage.conf**) - **status** is *Rejected*	* Step 1: (Message: **TriggerMessage.req**) - **requestMessage** is *MeterValues* - **connectorId** is configured *NumberOfConnectors + 1*
Expected result(s) / behaviour	The Charge Point does not send the message that was requested by the Central System.	n/a

2.22. SmartCharging

2.22.1. Central Smart Charging

Central Smart Charging - TxDefaultProfile

Table 77. Test Case Id: TC_056_CS

Test case name	**Central Smart Charging - TxDefaultProfile**	
Test case Id	TC_056_CS	
Description	The Central System sets a default schedule for new transactions.	
Purpose	To check whether the Charge Point is able to handle a default schedule for new transactions.	
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SetChargingProfile.conf**	**1.** The Central System sends a **SetChargingProfile.req**
	4. The Charge Point responds with a **GetCompositeSchedule.conf**	**3.** The Central System sends a **GetCompositeSchedule.req**

Test case name	Central Smart Charging - TxDefaultProfile	
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) - **status** is *Accepted* * Step 4: (Message: **GetCompositeSchedule.conf**) - **status** should be *Accepted* - **connectorId** should be *<Configured ConnectorId>* - The **chargingSchedule** fields: - **duration** should be *<Configured duration>* - **chargingRateUnit** should be *<Configured Charging Rate Unit>* - Between **startSchedule** and the current time should be equal or fewer seconds than *<Configured Max Time Deviation>* - **chargingSchedulePeriod** should be calculated accordingly.	* Step 1: (Message: **SetChargingProfile.req**) - **connectorId** *<Configured connectorId>* - **csChargingProfiles.stackLevel** *<Configured StackLevel>* - **csChargingProfiles.chargingProfilePurpose** *TxDefaultProfile* - **csChargingProfiles.chargingProfileKind** *Absolute* - **csChargingProfiles.chargingSchedule.duration** *<Configured duration>* - **csChargingProfiles.chargingSchedule.chargingRateUnit** *<Configured chargingRateUnit>* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].startPeriod** *0* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].limit** *if unit is A then 6(A) else 6000(W)* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].numberPhases** *<Configured numberPhases>* * Step 3: (Message: **GetCompositeSchedule.req**) - **connectorId** is *<Configured ConnectorId>* - **duration** is *<Configured duration>* - **chargingRateUnit** is *<Configured Charging Rate Unit>*
Expected result(s) / behaviour	n/a	n/a

Central Smart Charging - TxProfile

Table 78. Test Case Id: TC_057_CS

Test case name	Central Smart Charging - TxProfile	
Test case Id	TC_057_CS	
Description	The Central System sets a schedule for a running transaction.	
Purpose	To check whether the Charge Point is able to handle a Charging Profile with purpose TxProfile.	
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SetChargingProfile.conf**	**1.** The Central System sends a **SetChargingProfile.req**
	4. The Charge Point responds with a **GetCompositeSchedule.conf**	**3.** The Central System sends a **GetCompositeSchedule.req**

Test case name	**Central Smart Charging - TxProfile**	
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) - **status** is *Accepted* * Step 4: (Message: **GetCompositeSchedule.conf**) - **status** should be *Accepted* - **connectorId** should be *<Configured ConnectorId>* - The **chargingSchedule** fields: - **duration** should be *<Configured duration>* - **chargingRateUnit** should be *<Configured Charging Rate Unit>* - Between **startSchedule** and the current time should be equal or fewer seconds than *<Configured Max Time Deviation>* - **chargingSchedulePeriod** should be calculated accordingly.	* Step 1: (Message: **SetChargingProfile.req**) - **connectorId** *<Configured connectorId>* - **csChargingProfiles.transactionId** is *<transactionId returned by Charging Station before>* - **csChargingProfiles.stackLevel** *<Configured StackLevel>* - **csChargingProfiles.chargingProfilePurpose** is *TxProfile* - **csChargingProfiles.chargingProfileKind** is *Absolute* - **csChargingProfiles.chargingSchedule.chargingRateUnit** *<Configured chargingRateUnit>* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].startPeriod** *0* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].limit** *if unit is A then 6(A) else 6000(W)* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].numberPhases** *<Configured numberPhases>* * Step 3: (Message: **GetCompositeSchedule.req**) - **connectorId** is *<Configured ConnectorId>* - **duration** is *<Configured duration>* - **chargingRateUnit** is *<Configured Charging Rate Unit>*
Expected result(s) / behaviour	n/a	n/a

Central Smart Charging - No ongoing transaction

Table 79. Test Case Id: TC_058_1_CS

Test case name	**Central Smart Charging - No ongoing transaction**	
Test case Id	TC_058_1_CS	
Description	The Central System sets a schedule for a transaction (that is not running).	
Purpose	To check whether a Charge Point is able to reject a schedule with the wrong *ChargingProfilePurpose*	
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile and no transaction is running.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SetChargingProfile.conf** message OR with a CallError.	**1.** The Central System sends a **SetChargingProfile.req** message with a *connectorId* and a *ChargingProfile* that includes a *transactionId* and a *ChargingProfilePurpose*

Test case name	**Central Smart Charging - No ongoing transaction**	
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) **status** is *Rejected* OR (Message: **CallError** **ErrorCode** is *PropertyConstraintViolation*.	* Step 1: (Message: **SetChargingProfile.req**) **ChargingProfilePurpose** is *TxProfile*
Expected result(s) / behaviour	The Charge Point rejects the **SetChargingProfile.req** message.	n/a.

Central Smart Charging - Wrong transactionId

Table 80. Test Case Id: TC_058_2_CS

Test case name	**Central Smart Charging - Wrong transactionId**	
Test case Id	TC_058_2_CS	
Description	The Central System sets a schedule for a transaction (that is not running).	
Purpose	To check whether a Charge Point is able to reject a schedule with the wrong *ChargingProfilePurpose*	
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile and no transaction is running.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SetChargingProfile.conf** message.	**1.** The Central System sends a **SetChargingProfile.req** message with a *connectorId* and a *ChargingProfile* that includes a *transactionId* and a *ChargingProfilePurpose* .
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) **status** is *Rejected*	* Step 1: (Message: **SetChargingProfile.req**) The **ChargingProfilePurpose** is *TxProfile* The **connectorId** equals the **connectorId** from step 5 (and is > 0). The **transactionId** does NOT equal the **transactionId** from step 6.
Expected result(s) / behaviour	The Charge Point rejects the **SetChargingProfile.req** message.	n/a.

Central Smart Charging - TxDefaultProfile - with ongoing transaction

Table 81. Test Case Id: TC_082_CS

Test case name	**Central Smart Charging - TxDefaultProfile - with ongoing transaction**
Test case Id	TC_082_CS
Description	The Central System sets a default schedule for a currently ongoing transaction.
Purpose	To check whether the Charge Point is able to handle a default schedule for a currently ongoing transaction.
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): - *Charging*

Test case name	**Central Smart Charging - TxDefaultProfile - with ongoing transaction**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SetChargingProfile.conf**	**1.** The Central System sends a **SetChargingProfile.req**
	4. The Charge Point responds with a **GetCompositeSchedule.conf**	**3.** The Central System sends a **GetCompositeSchedule.req**
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) - **status** is *Accepted* * Step 4: (Message: **GetCompositeSchedule.conf**) - **status** should be *Accepted* - **connectorId** should be *<Configured ConnectorId>* - The **chargingSchedule** fields: - **duration** should be *<Configured duration>* - **chargingRateUnit** should be *<Configured Charging Rate Unit>* - Between **startSchedule** and the current time should be equal or fewer seconds than *<Configured Max Time Deviation>* - **chargingSchedulePeriod** should be calculated accordingly.	* Step 1: (Message: **SetChargingProfile.req**) - **connectorId** *<Configured connectorId>* - **csChargingProfiles.stackLevel** *<Configured StackLevel>* - **csChargingProfiles.chargingProfilePurpose** *TxDefaultProfile* - **csChargingProfiles.chargingProfileKind** *Absolute* - **csChargingProfiles.chargingSchedule.duration** *<Configured duration>* - **csChargingProfiles.chargingSchedule.chargingRateUnit** *<Configured chargingRateUnit>* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].startPeriod** *0* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].limit** *if unit is A then 6(A) else 6000(W)* - **csChargingProfiles.chargingSchedule.chargingSchedulePeriod[0].numberPhases** *<Configured numberPhases>* * Step 3: (Message: **GetCompositeSchedule.req**) - **connectorId** is *<Configured ConnectorId>* - **duration** is *<Configured duration>* - **chargingRateUnit** is *<Configured Charging Rate Unit>*
Expected result(s) / behaviour	n/a	n/a

2.22.2. Get Composite Schedule

Table 82. Test Case Id: TC_066_CS

Test case name	**Get Composite Schedule**
Test case Id	TC_066_CS
Description	The Central System sends 3 *ChargingProfiles* to a Charge Point and then requests (and validates) the composite schedule.
Purpose	To check whether the Charge Point is able to handle *ChargingProfilePurposes* as specified in OCPP.
Prerequisite(s)	- The Charge Point supports the Smart Charging feature profile. - Configuration key **MaxChargingProfilesInstalled** is >= *3*. - Configuration key **ChargingScheduleMaxPeriods** is >= *5*.

Test case name	**Get Composite Schedule**	
Before	**Configuration State(s):** n/a	
	Memory State(s): *SetChargingProfile* with ChargingProfile 1: **chargingProfilePurpose** is *ChargingStationMaxProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *0* **connectorId** *0* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *86400* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0*, **limit** *10*	
	ChargingProfile 2: **chargingProfilePurpose** is *TxDefaultProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *0* **connectorId** *<Configured ConnectorId>* **validFrom** *<current dateTime - <Configured max time deviation> seconds>* **validTo** *<current dateTime - <Configured max time deviation> + 401 seconds>* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *300* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0,60,120,180,260*, **limit** *6,10,8,15,8*	ChargingProfile 3: **chargingProfilePurpose** is *TxProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *0* **connectorId** *<Configured ConnectorId>* **validFrom** *<current dateTime - <Configured max time deviation> seconds>* **validTo** *<current dateTime - <Configured max time deviation> + 401 seconds>* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *260* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0,50,140,200,240*, **limit** *8,11,16,6,12*
	Reusable State(s): - *Charging* **Note:** The reusable state *Charging* is executed before the memory states *SetChargingProfile*.	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetCompositeSchedule.conf**	**1.** The Central System sends a **GetCompositeSchedule.req**

Test case name	**Get Composite Schedule**	
Tool validation(s)	* Step 2: (Message: **GetCompositeSchedule.conf**) **status** *Accepted* **connectorId** is *<Configured ConnectorId>* **ChargingSchedule**: **duration** *400* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the* ***limit*** *field will be multiplied by 1000.* *Note: The period of time between sending the second SetChargingProfile.req and the* ***scheduleStart*** *from the GetCompositeSchedule.conf is called* ***x****:* **startPeriod** *0*, **limit** *8* **startPeriod** *(50 - x)*, **limit** *10* **startPeriod** *(200 - x)*, **limit** *6* **startPeriod** *(240 - x)*, **limit** *10*	* Step 1: (Message: **GetCompositeSchedule.req**) - **connectorId** is *<Configured ConnectorId>* - **duration** is *400* - **chargingRateUnit** is *<Configured Charging Rate Unit>*
Expected result(s) / behaviour	The Charge Point is able to combine different *ChargingProfiles* from the Central System and return a composite schedule.	n/a

2.22.3. Clear Charging Profile

Table 83. Test Case Id: TC_067_CS

Test case name	**Clear Charging Profile**
Test case Id	TC_067_CS
Description	The Central Systems sets charging profiles and clears it.
Purpose	To check whether the Charge Point is able to clear charging profiles.
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile.

Test case name	**Clear Charging Profile**	
Before	**Configuration State(s):** n/a	
	Memory State(s): *SetChargingProfile* with ChargingProfile 1: **chargingProfilePurpose** is *TxDefaultProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *1* **connectorId** *<Configured connectorId>* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *400* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0,60,200*, **limit** *6,8,10*	
	ChargingProfile 2: **chargingProfilePurpose** is *TxDefaultProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *0* **connectorId** *<Configured ConnectorId>* **validFrom** *<current dateTime - <Configured max time deviation> seconds>* **validTo** *<current dateTime - <Configured max time deviation> + 401 seconds>* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *400* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0,100*, **limit** *7,9*	ChargingProfile 3: **chargingProfilePurpose** is *ChargePointMaxProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *0* **connectorId** *0* **validFrom** *<current dateTime - <Configured max time deviation> seconds>* **validTo** *<current dateTime - <Configured max time deviation> + 401 seconds>* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *86400* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0,200*, **limit** *11,12*
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ClearChargingProfile.conf**	**1.** The Central System sends a **ClearChargingProfile.req**
	4. The Charge Point responds with a **GetCompositeSchedule.conf**	**3.** The Central System sends a **GetCompositeSchedule.req**
	6. The Charge Point responds with a **ClearChargingProfile.conf**	**5.** The Central System sends a **ClearChargingProfile.req**
	8. The Charge Point responds with a **GetCompositeSchedule.conf**	**7.** The Central System sends a **GetCompositeSchedule.req**
	10. The Charge Point responds with a **ClearChargingProfile.conf**	**9.** The Central System sends a **ClearChargingProfile.req**
	12. The Charge Point responds with a **GetCompositeSchedule.conf**	**11.** The Central System sends a **GetCompositeSchedule.req**

Test case name	**Clear Charging Profile**	
Tool validation(s)	* Step 2 / 6 / 10: (Message: **ClearChargingProfile.conf**) - The **status** is *Accepted* * Step 4: (Message: **GetCompositeSchedule.conf**) **status** *Accepted* **connectorId** is *<Configured ConnectorId>* **ChargingSchedule**: **duration** *350* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* *Note: The period of time between sending the second SetChargingProfile.req and the **scheduleStart** from the GetCompositeSchedule.conf is called **x**:* **startPeriod** *0*, **limit** *7* **startPeriod** *(100 - x)*, **limit** *9* * Step 8: (Message: **GetCompositeSchedule.conf**) **status** *Accepted* **connectorId** is *<Configured ConnectorId>* **ChargingSchedule**: **duration** *350* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* *Note: The period of time between sending the second SetChargingProfile.req and the **scheduleStart** from the GetCompositeSchedule.conf is called **x**:* **startPeriod** *0*, **limit** *11* **startPeriod** *(200 - x)*, **limit** *12* * Step 12: (Message: **GetCompositeSchedule.conf**) **status** *Accepted* **connectorId** is *<Configured ConnectorId>* **ChargingSchedule**: **duration** *350* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* *Note: The period of time between sending the second SetChargingProfile.req and the **scheduleStart** from the GetCompositeSchedule.conf is called **x**:* **startPeriod** *0*, **limit** *<The local limit of the Charging Station>*	* Step 1: (Message: **ClearChargingProfile.req**) - The **id** is the chargingProfileId from the first *ChargingProfile*. - All other fields are omitted. * Step 5: (Message: **ClearChargingProfile.req**) - The **chargingProfilePurpose** is the purpose from the second *ChargingProfile*. - The **stackLevel** is the stackLevel from the second *ChargingProfile*. - All other fields are omitted. * Step 9: (Message: **ClearChargingProfile.req**) - All fields are omitted.
Expected result(s) / behaviour	n/a	n/a

2.22.4. Stacking Charging Profiles

Table 84. Test Case Id: TC_072_CS

Test case name	**Stacking Charging Profiles**	
Test case Id	TC_072_CS	
Description	The Central System sends 2 *ChargingProfiles* to a Charge Point and then requests (and validates) the composite schedule.	
Purpose	To check whether the Charge Point is able to stack *ChargingProfiles* as specified in OCPP.	
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): *SetChargingProfile* with ChargingProfile 1: **chargingProfilePurpose** is *TxDefaultProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *0* **connectorId** *<Configured ConnectorId>* **validFrom** *<current dateTime - <Configured max time deviation> seconds>* **validTo** *<current dateTime - <Configured max time deviation> + 401 seconds>* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *400* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0*, **limit** *6* **startPeriod** *100*, **limit** *8* **startPeriod** *200*, **limit** *10* ChargingProfile 2: **chargingProfilePurpose** is *TxDefaultProfile* **chargingProfileKind** should be *Absolute* **stackLevel** should be *1* **connectorId** *<Configured ConnectorId>* **validFrom** *<current dateTime - <Configured max time deviation> seconds>* **validTo** *<current dateTime - <Configured max time deviation> + 401 seconds>* **startSchedule** *<current dateTime - <Configured max time deviation> seconds>* **numberPhases** *<Configured numberPhases>* **ChargingSchedule**: **duration** *150* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* **startPeriod** *0*, **limit** *7* **startPeriod** *100*, **limit** *9*	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetCompositeSchedule.conf**	**1.** The Central System sends a **GetCompositeSchedule.req**

Test case name	**Stacking Charging Profiles**	
Tool validation(s)	* Step 2: (Message: **GetCompositeSchedule.conf**) **status** *Accepted* **connectorId** is *<Configured ConnectorId>* **ChargingSchedule**: **duration** *350* **chargingRateUnit** *<Configured chargingRateUnit>* *Note: If <Configured chargingRateUnit> is W, then the **limit** field will be multiplied by 1000.* *Note: The period of time between sending the second SetChargingProfileRequest and the **scheduleStart** from the GetCompositeScheduleResponse is called **x**:* ***startPeriod** _0*, **limit** *7* **startPeriod** *(100 - x)*, **limit** *9* **startPeriod** *(150 - x)*, **limit** *8* **startPeriod** *(200 - x)*, **limit** *10*	* Step 1: (Message: **GetCompositeSchedule.req**) - **connectorId** is *<Configured ConnectorId>* - **duration** is *350* - **chargingRateUnit** is *<Configured Charging Rate Unit>*
Expected result(s) / behaviour	The Charge Point is able to stack multiple *ChargingProfiles* from the Central System and return a composite schedule.	n/a

2.22.5. Remote Start Transaction with Charging Profile

Remote Start Transaction with Charging Profile

Table 85. Test Case Id: TC_059_CS

Test case name	**Remote Start Transaction with Charging Profile**	
Test case Id	TC_059_CS	
Description	The Central System starts a transaction on a Charge Point with a *ChargingProfile*	
Purpose	To check whether the Charge Point is able to start a transaction with a Charging Profile initiated from the Central System.	
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	1. Execute *Charging* with a *ChargingProfile* with purpose *TxProfile*.	
	3. The Charge Point responds with a **GetCompositeSchedule.conf**	**2.** The Central System sends a **GetCompositeSchedule.req**

Test case name	Remote Start Transaction with Charging Profile	
Tool validation(s)	* Step 3: (Message: **GetCompositeSchedule.conf**) - **status** should be *Accepted* - **connectorId** should be *<Configured ConnectorId>* - The **chargingSchedule** fields: - **duration** should be *<Configured duration>* - **chargingRateUnit** should be *<Configured Charging Rate Unit>* - Between **startSchedule** and the current time should be equal or fewer seconds than *<Configured Max Time Deviation>* - **chargingSchedulePeriod** should be calculated accordingly.	* Step 2: (Message: **GetCompositeSchedule.req**) - **connectorId** is *<Configured ConnectorId>* - **duration** is *<Configured duration>* - **chargingRateUnit** is *<Configured Charging Rate Unit>*
Expected result(s) / behaviour	A transaction is started on the Charge Point and the profile sent by the Central System is followed by the Charge Point.	n/a

Remote Start Transaction with Charging Profile - Rejected

Table 86. Test Case Id: TC_060_CS

Test case name	Remote Start Transaction with Charging Profile - Rejected	
Test case Id	TC_060_CS	
Description	The Central System tries to start a transaction on a Charge Point but this is rejected.	
Purpose	To check whether the Charge Point is able to reject a a transaction with a Charging Profile initiated from the Central System.	
Prerequisite(s)	The Charge Point supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **RemoteStartTransaction.conf** OR with a CallError.	**1.** The Central Systems sends a **RemoteStartTransaction.req** message to the Charge Point.
Tool validation(s)	* Step 2: (Message: **RemoteStartTransaction.conf**) The **status** is *Rejected* OR (Message: **CallError** **ErrorCode** is *PropertyConstraintViolation*.	* Step 1: (Message: **RemoteStartTransaction.req**) The **ChargingProfile.chargingProfilePurpose** is NOT *TxProfile*
Expected result(s) / behaviour	n/a	n/a

2.23. DataTransfer

2.23.1. Data Transfer to a Charge Point

Table 87. Test Case Id: TC_062_CS

Test case name	**Data Transfer to a Charge Point**	
Test case Id	TC_062_CS	
Description	The Central System sends a vendor specific message to a Charge Point.	
Purpose	To check whether the Charge Point can reject vendor specific messages.	
Prerequisite(s)	The Charge Point does not support DataTransfer for a specific *vendorId*.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **DataTransfer.conf** message.	**1.** The Central System sends a **DataTransfer.req** message with a specific *vendorId* to the Charge Point.
Tool validation(s)	* Step 2: (Message: **DataTransfer.conf**) The **status** is *Rejected* OR *UnknownMessageId* OR *UnknownVendorId* **Note:** The **status** *Accepted* is allowed, but the vendor should be warned about this behaviour.	n/a
Expected result(s) / behaviour	The Charge Point does not accept the **DataTransfer.req** message.	n/a

2.24. Security

2.24.1. Secure connection setup

Update Charge Point Password for HTTP Basic Authentication

Table 88. Test Case Id: TC_073_CS

Test case name	**Update Charge Point Password for HTTP Basic Authentication**	
Test case Id	TC_073_CS	
Description	The Central System can configure a new password for HTTP Basic Authentication, the Central System can send a new value for the BasicAuthPassword Configuration key.	
Purpose	To check if the Charge Point is able to switch to a new Basic Authentication password.	
Prerequisite(s)	The Charge Point supports Security profile 1 and/or 2.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
	3. The Charge Point disconnects its current connection and reconnects to the Central System with the new password.	

Test case name	**Update Charge Point Password for HTTP Basic Authentication**	
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) **status** is *Accepted* * Step 3: *The Charge Point reconnects to the Central System with the new password.*	* Step 1: (Message: **ChangeConfiguration.req**) **key** is *AuthorizationKey* **value** is *4F43415F4F4354545F61646D696E5F74657374* ("OCA_OCTT_admin_test" HexEncoded)
Expected result(s) / behaviour	n/a	n/a

Update Charge Point Certificate by request of Central System

Table 89. Test Case Id: TC_074_CS

Test case name	**Update Charge Point Certificate by request of Central System**	
Test case Id	TC_074_CS	
Description	The tool shall take on the role of both Central System and Certificate Authority Server. Which means it will sign the certificate with its own certificate.	
Purpose	To test if the Charge Point renews its ChargePointCertificate when the Central System requests to do so.	
Prerequisite(s)	The Charge Point supports security profile 3.	
Before	**Configuration State(s):** - *CpoName* is *<The configured Vendor Name>*.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ExtendedTriggerMessage.conf**	**1.** The Central System sends a **ExtendedTriggerMessage.req**
	[The Charge Point generates a new public/private key pair and generates a Certificate Signing Request.] **3.** The Charge Point sends a **SignCertificate.req**.	**4.** The Central System responds with a **SignCertificate.conf**.
	[The Charge Point verifies the validity of the signed certificate.] **6.** The Charge Point responds with a **CertificateSigned.conf**.	[Certificate Authority Server signs the certificate.] **5.** The Central System sends a **CertificateSigned.req**.
	7. The Charge Point disconnects its current connection and reconnects to the Central System with the new certificate.	
Tool validation(s)	* Step 2: (Message: **ExtendedTriggerMessage.conf**) The **status** is *Accepted* * Step 6: (Message: **CertificateSigned.conf**) The **status** is *Accepted* * Step 7: *The Charge Point reconnects to the Central System with the new certificate.*	* Step 1: (Message: **ExtendedTriggerMessage.req**) The **requestedMessage** is *SignChargePointCertificate* The **connectorId** is *<Omitted>* * Step 4: (Message: **SignCertificate.conf**) The **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

Install a certificate on the Charge Point

Table 90. Test Case Id: TC_075_1_CS

Test case name	**Install a certificate on the Charge Point - ManufacturerRootCertificate**
Test case Id	TC_075_1_CS

Test case name	Install a certificate on the Charge Point - ManufacturerRootCertificate	
Description	The Central System requests the Charge Point to install a new manufacturer root certificate.	
Purpose	To check if the Charge Point is able to install a certificate.	
Prerequisite(s)	- The Charge Point supports Security profile 2 and/or 3. - The tester configured the root certificate in the store.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **InstallCertificate.conf**	**1.** The Central System sends a **InstallCertificate.req**
	4. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**3.** The Central System sends a **GetInstalledCertificateIds.req**
Tool validation(s)	* Step 2: (Message: **InstallCertificate.conf**) **status** is *Accepted* * Step 4: (Message: **GetInstalledCertificateIds.conf**) The **status** is *Accepted* **certificateHashData** is *<Includes the certificate information of the installed certificate from step 1.>* The OCTT verifies that the certificate is present, based on its own calculation of the **certificateHashData**.	* Step 1: (Message: **InstallCertificate.req**) **certificateType** is *ManufacturerRootCertificate* **certificate** is *<Certificate from the store>* * Step 3: (Message: **GetInstalledCertificateIds.req**) The **certificateType** is *ManufacturerRootCertificate*
Expected result(s) / behaviour	n/a	n/a

Table 91. Test Case Id: TC_075_2_CS

Test case name	Install a certificate on the Charge Point - CentralSystemRootCertificate	
Test case Id	TC_075_2_CS	
Description	The Central System requests the Charge Point to install a new Central System root certificate.	
Purpose	To check if the Charge Point is able to install a certificate.	
Prerequisite(s)	- The Charge Point supports Security profile 2 and/or 3. - The tester configured the root certificate in the store.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **InstallCertificate.conf**	**1.** The Central System sends a **InstallCertificate.req**
	4. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**3.** The Central System sends a **GetInstalledCertificateIds.req**

Test case name	**Install a certificate on the Charge Point - CentralSystemRootCertificate**	
Tool validation(s)	* Step 2: (Message: **InstallCertificate.conf**) **status** is *Accepted* * Step 4: (Message: **GetInstalledCertificateIds.conf**) The **status** is *Accepted* **certificateHashData** is *<Includes the certificate information of the installed certificate from step 1.>* The OCTT verifies that the certificate is present, based on its own calculation of the **certificateHashData**.	* Step 1: (Message: **InstallCertificate.req**) **certificateType** is *CentralSystemRootCertificate* **certificate** is *<Certificate from the store>* * Step 3: (Message: **GetInstalledCertificateIds.req**) The **certificateType** is *CentralSystemRootCertificate*
Expected result(s) / behaviour	n/a	n/a

Delete a specific certificate from the Charge Point

Table 92. Test Case Id: TC_076_CS

Test case name	**Delete a specific certificate from the Charge Point**	
Test case Id	TC_076_CS	
Description	To facilitate the management of the Charge Point's installed certificates, a method of deleting an installed certificate is provided. The Central System requests the Charge Point to delete a specific certificate.	
Purpose	To check if the Charge Point is able to delete an installed certificate.	
Prerequisite(s)	- The Charge Point supports Security profile 2 and/or 3.	
Before	**Configuration State(s):** n/a	
	Memory State(s): - *CertificateInstalled*	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**1.** The Central System sends a **GetInstalledCertificateIds.req**
	4. The Charge Point responds with a **DeleteCertificate.conf**	**3.** The Central System sends a **DeleteCertificate.req**
	6. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**5.** The Central System sends a **GetInstalledCertificateIds.req**
Tool validation(s)	* Step 4: (Message: **DeleteCertificate.conf**) **status** is *Accepted* * Step 6: (Message: **GetInstalledCertificateIds.conf**) **certificateHashData** *<Does not include the certificate information of the removed certificate.>* The OCTT verifies that the certificate is removed, based on its own calculation of the **certificateHashData**.	* Step 3: (Message: **DeleteCertificate.req**) **certificateHashData** is *<Includes the certificate information of the certificate from the configured CentralSystemRootCertificate, using the hashAlgorithm provided at step 2.>* * Step 5: (Message: **GetInstalledCertificateIds.req**) The **certificateType** is *<Equals the type of the removed certificate.>*
Expected result(s) / behaviour	n/a	n/a

2.24.2. Security event/logging

Invalid ChargePointCertificate Security Event

Table 93. Test Case Id: TC_077_CS

Test case name	**Invalid ChargePointCertificate Security Event**	
Test case Id	TC_077_CS	
Description	The Charge Point notifies the Central System of an invalid certificate. The tool shall take on the role of both Central System and Certificate Authority Server. Which means it will sign the certificate using its own certificate.	
Purpose	To check if the Charge Point is able to register a security event and is able not notify the Central System about it.	
Prerequisite(s)	The Charge Point supports security profile 3.	
Before	**Configuration State(s):** - *CpoName* is *<The configured Vendor Name>*.	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ExtendedTriggerMessage.conf**	**1.** The Central System sends a **ExtendedTriggerMessage.req**
	[The Charge Point generates a new public/private key pair and generates a Certificate Signing Request.] **3.** The Charge Point sends a **SignCertificate.req**.	**4.** The Central System responds with a **SignCertificate.conf**.
	[The Charge Point verifies the validity of the signed certificate.] **6.** The Charge Point responds with a **CertificateSigned.conf**.	**5.** The Central System sends a **CertificateSigned.req**.
	7. The Charge Point sends a **SecurityEventNotification.req**	**8.** The Central System responds with a **SecurityEventNotification.conf**
Tool validation(s)	* Step 2: (Message: **ExtendedTriggerMessage.conf**) The **status** is *Accepted* * Step 6: (Message: **CertificateSigned.conf**) The **status** is *Rejected* * Step 7: (Message: **SecurityEventNotification.req**) The **type** is *InvalidChargePointCertificate*	* Step 1: (Message: **ExtendedTriggerMessage.req**) The **requestedMessage** is *SignChargePointCertificate* The **connectorId** is *<Omitted>* * Step 4: (Message: **SignCertificate.conf**) The **status** is *Accepted* * Step 5: (Message: **CertificateSigned.req**) The **certificate** is *<An invalid certificate>*
Expected result(s) / behaviour	n/a	n/a

Invalid CentralSystemCertificate Security Event

Table 94. Test Case Id: TC_078_CS

Test case name	**Invalid CentralSystemCertificate Security Event**
Test case Id	TC_078_CS
Description	The Charge Point notifies the Central System of an invalid certificate.
Purpose	To check if the Charge Point is able to register a security event and is able not notify the Central System about it.
Prerequisite(s)	The Charge Point supports Security profile 2 and/or 3.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Invalid CentralSystemCertificate Security Event**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with an **InstallCertificate.conf**	**1.** The Central System sends an **InstallCertificate.req**
	3. The Charge Point sends a **SecurityEventNotification.req**	**4.** The Central System responds with a **SecurityEventNotification.conf**
Tool validation(s)	* Step 2: (Message: **InstallCertificate.conf**) **status** is *Rejected* * Step 3: (Message: **SecurityEventNotification.req**) The **type** is *InvalidCentralSystemCertificate*	* Step 1: (Message: **InstallCertificate.req**) **certificateType** is *CentralSystemRootCertificate* **certificate** is *<An invalid/expired certificate>*
Expected result(s) / behaviour	n/a	n/a

Get Security Log

Table 95. Test Case Id: TC_079_CS

Test case name	**Get Security Log**	
Test case Id	TC_079_CS	
Description	The Charge Point uploads a security log to a specified location based on a request of the Central System.	
Purpose	To check whether the Charge Point can upload its security log.	
Prerequisite(s)	The Charge Point supports a security profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetLog.conf**.	**1.** The Central System sends a **GetLog.req**.
	[The Charge Point starts uploading the security log.] **3.** The Charge Point sends a **LogStatusNotification.req**.	**4.** The Central System responds with a **LogStatusNotification.conf**.
	[The Charge Point has finished uploading the security log.] **5.** The Charge Point sends a **LogStatusNotification.req**.	**6.** The Central System responds with a **LogStatusNotification.conf**.
Tool validation(s)	* Step 2: (Message: **GetLog.conf**) The **status** is *Accepted* * Step 3: (Message: **LogStatusNotification.req**) The **status** is *Uploading* * Step 5: (Message: **LogStatusNotification.req**) The **status** is *Uploaded*	* Step 1: (Message: **GetLog.req**) The **log.remoteLocation** is *<Configured log location>* The **logType** is *SecurityLog*
Expected result(s) / behaviour	The Charge Point has uploaded the security log to the **log.remoteLocation** that was sent in step 1.	n/a

2.24.3. Secure firmware update

Secure Firmware Update

Table 96. Test Case Id: TC_080_CS

Test case name	**Secure Firmware Update**	
Test case Id	TC_080_CS	
Description	The firmware of a Charge Point is updated in a secure way.	
Purpose	To check whether the Charge Point can update its firmware in a secure way.	
Prerequisite(s)	- The Charge Point supports the FirmwareManagement feature profile AND - The Charge Point supports a security profile AND - A firmware is prepared on a server (For example ftp) AND - The tester configured the signature calculated over the firmware at the 'Signature' test data field.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[The Charge Point has verified the certificate] **2.** The Charge Point responds with a **SignedUpdateFirmware.conf**.	**1.** The Central System sends a **SignedUpdateFirmware.req**.
	[The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**4.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**6.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.
	[The Charge Point has verified the signature] **7.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**8.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.
	[Before installing the firmware, the Charge Point MAY set all connectors to Unavailable. If the Charge Point supports installation of firmware during a charging session, the Charge Point MAY install the firmware after only setting all other connectors to Unavailable.] [The Charge Point starts installing the firmware] **9.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**10.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.
	11. The Charge Point sends a **BootNotification.req**.	**12.** The Central System responds with a **BootNotification.conf**.
	13. The Charge Point optionally sends a **SecurityEventNotification.req** With **type** *FirmwareUpdated*	**14.** The Central System responds with a **SecurityEventNotification.conf**
	[On all connectors and connector = 0] **15.** The Charge Point sends a **StatusNotification.req**.	**16.** The Central System responds with a **StatusNotification.conf**.
	[The Charge Point has finished installing the firmware] **17.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**18.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.

Test case name	**Secure Firmware Update**	
Tool validation(s)	* Step 2: (Message: **SignedUpdateFirmware.conf**) The **status** is *Accepted* * Step 3: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *SignatureVerified* * Step 9: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Installing* * Step 15: (Message: **StatusNotification.req**) The **status** is *Available* * Step 17: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Installed* * Step 11 / 13 / 15 / 17: The messages can be in a different order.	* Step 1: (Message: **SignedUpdateFirmware.req**) The **firmware.location** is *<Configured firmware location>* * Step 14: (Message: **BootNotification.conf**) The **status** is *Accepted*
Expected result(s) / behaviour	The Charge Point handles the firmware update correctly and is Available after the update.	n/a

Secure Firmware Update - Invalid Signature

Table 97. Test Case Id: TC_081_CS

Test case name	**Secure Firmware Update - Invalid Signature**
Test case Id	TC_081_CS
Description	The Charge Point validates the Signature and deems it invalid.
Purpose	To check whether the Charge Point validates the signature.
Prerequisite(s)	- The Charge Point supports the FirmwareManagement feature profile AND - The Charge Point supports a security profile AND - A firmware is prepared on a server (For example ftp) AND - The tester configured the signature calculated over the firmware at the 'Signature' test data field.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Secure Firmware Update - Invalid Signature**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SignedUpdateFirmware.conf**.	**1.** The Central System sends a **SignedUpdateFirmware.req**.
	[The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**4.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**6.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.
	[The Charge Point verifies the signature and deems it invalid] **7.** The Charge Point sends a **SignedFirmwareStatusNotification.req**.	**8.** The Central System responds with a **SignedFirmwareStatusNotification.conf**.
Tool validation(s)	* Step 2: (Message: **SignedUpdateFirmware.conf**) The **status** is *Accepted* * Step 3: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *InvalidSignature*	* Step 1: (Message: **SignedUpdateFirmware.req**) The **firmware.location** is *<Configured firmware location>* The **firmware.signature** is *<An invalid signature.>*
Expected result(s) / behaviour	The Charge Point rejects the firmware, because of an invalid signature.	n/a

Upgrade security profile

Table 98. Test Case Id: TC_083_CS

Test case name	**Upgrade security profile**
Test case Id	TC_083_CS
Description	The Central System can upgrade the connection using a higher Security Profile, the Central System can send a new value for the SecurityProfile Configuration key.
Purpose	To check if the Charge Point is able to upgrade the Security Profile.
Prerequisite(s)	The Charge Point is connected with *SecurityProfile* 1 or 2.
Before	**Configuration State(s):** n/a
	Memory State(s): - *CertificateInstalled* if *SecurityProfile* is 1. - RenewChargePointCertificate if *SecurityProfile* is 2.
	Reusable State(s): n/a

Test case name	**Upgrade security profile**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
	4. The Charge Point responds with a **Reset.conf**	**3.** The Central System sends a **Reset.req**
	5. The Charge Point sends a **BootNotification.req**	**6.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
	Note: Steps 3-8 will only be executed if *RebootRequired* or Charge Point does not reconnect itself.	
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) - **status** should be *Accepted* or *RebootRequired* * Step 4: (Message: **Reset.conf**) - **status** should be *Accepted* * Step 7: (Message: **StatusNotification.req**) - **status** should be *Available*	* Step 1: (Message: **ChangeConfiguration.req**) - **key** is *SecurityProfile* - **value** is *<One level higher than the configured security profile>* * Step 3: (Message: **Reset.req**) - **type** is *Hard* * Step 6: (Message: **BootNotification.conf**) - **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

Downgrade security profile - Rejected

Table 99. Test Case Id: TC_084_CS

Test case name	**Downgrade security profile - Rejected**	
Test case Id	TC_084_CS	
Description	The Central System can upgrade the connection using a higher Security Profile. It is not possible to downgrade to a lower Security Profile.	
Purpose	To check if the Charge Point rejects downgrading the Security Profile.	
Prerequisite(s)	The Charge Point is connected with *SecurityProfile* 2 or 3.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) - **status** is *Rejected*	* Step 1: (Message: **ChangeConfiguration.req**) - **key** is *SecurityProfile* - **value** is *<One level lower than the configured security profile.>*
Expected result(s) / behaviour	n/a	n/a

Basic Authentication - Valid username/password combination

Table 100. Test Case Id: TC_085_CS

Test case name	Basic Authentication - Valid username/password combination	
Test case Id	TC_085_CS	
Description	The Charge Point uses Basic authentication to authenticate itself to the Central System, when using security profile 1 or 2.	
Purpose	To verify whether the Charge Point is able to authenticate itself to the Central System using Basic Authentication.	
Prerequisite(s)	The Charge Point supports security profile 1 and/or 2.	
Before (Preparations)	**Configuration State:** N/a	
	Memory State: N/a	
	Reusable State(s): The Charge Point is triggered to reset.	
Main (Test scenario)	**Charge Point (SUT)**	**Central System (Tool)**
	1. The Charge Point sends a HTTP upgrade request to the Central System	**2.** The Central System upgrades the connection to a WebSocket connection.
	3. The Charge Point sends a **BootNotification.req**	**4.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validations	* Step 1: The authorization header of the HTTP upgrade request must be formatted as follows: *AUTHORIZATION: Basic <Base64 encoded(<ChargePointId>:<Configured AuthorizationKey>)>* - The ChargePointId, must equal the ChargePointId provided at the end of the connection url string of the HTTP request. - Hex encoded representation of the authorization key must consist of minimum 20 and maximum 40 characters. - The authorization key must consist of minimum 16 characters.	
	Post scenario validations: N/a	

TLS - server-side certificate - Valid certificate

Table 101. Test Case Id: TC_086_CS

Test case name	TLS - server-side certificate - Valid certificate
Test case Id	TC_086_CS
Description	The Central System uses a server-side certificate to identify itself to the Charge Point, when using security profile 2 or 3.
Purpose	To verify whether the Charge Point is able to receive a server certificate provided by the Central System and setup a secured WebSocket connection.
Prerequisite(s)	The Charge Point supports security profile 2 and/or 3.
Before (Preparations)	**Configuration State:** N/a
	Memory State: N/a
	Reusable State(s): The Charge Point is triggered to reset.

Test case name	TLS - server-side certificate - Valid certificate	
Main (Test scenario)	**Charge Point (SUT)**	**Central System (Tool)**
	1. The Charge Point initiates a TLS handshake and sends a Client Hello to the Central System.	**2.** The Central System responds with a Server Hello With the <Configured server certificate>
	3. The Charge Point performs the following actions: Send client certificate Client Key Exchange Certificate verify Change Cipher Spec Finished Note(s): *- The client certificate is only sent when the Charge Point uses security profile 3.*	**4.** The Central System performs the following actions: Change Cipher Spec Finished
	5. The Charge Point sends a HTTP upgrade request to the Central System Note(s): *- The HTTP request only contains a username/password combination when the Charge Point uses security profile 2.*	**6.** The Central System upgrades the connection to a (secured) WebSocket connection.
	7. The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
Tool validations	* Step 2: The OCTT validates the following before sending the server certificate: - The Charge Point must use TLS version 1.2 or above At least the following set of cipher suites must be supported: (TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 AND TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384) OR (TLS_RSA_WITH_AES_128_GCM_SHA256 AND TLS_RSA_WITH_AES_256_GCM_SHA384) * Step 5: The authorization header of the HTTP upgrade request must be formatted as follows: *AUTHORIZATION: Basic <Base64 encoded(<ChargePointId>:<Configured AuthorizationKey>)>* - The ChargePointId, must equal the ChargePointId provided at the end of the connection url string of the HTTP request. - Hex encoded representation of the authorization key must consist of minimum 20 and maximum 40 characters. - The authorization key must consist of minimum 16 characters.	
	Post scenario validations: N/a	

TLS - Client-side certificate - valid certificate

Table 102. Test Case Id: TC_087_CS

Test case name	TLS - Client-side certificate - valid certificate
Test case Id	TC_087_CS
Description	The Charge Point uses a client-side certificate to identify itself to the Central System, when using security profile 3.

Test case name	**TLS - Client-side certificate - valid certificate**	
Purpose	To verify whether the Charge Point is able to provide a valid client certificate and setup a secured WebSocket connection.	
Prerequisite(s)	The Charge Point supports security profile 3.	
Before (Preparations)	**Configuration State:** N/a	
	Memory State: N/a	
	Reusable State(s): The Charge Point is triggered to reset.	
Main (Test scenario)	**Charge Point (SUT)**	**Central System (Tool)**
	1. The Charge Point initiates a TLS handshake and sends a Client Hello to the Central System.	**2.** The Central System responds with a Server Hello With the <Configured server certificate>
	3. The Charge Point performs the following actions: Send client certificate Client Key Exchange Certificate verify Change Cipher Spec Finished	**4.** The Central System performs the following actions: Change Cipher Spec Finished
	5. The Charge Point sends a HTTP upgrade request to the Central System	**6.** The Central System upgrades the connection to a (secured) WebSocket connection.
	7. The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
Tool validations	* Step 4: The OCTT validates the following before finishing the TLS handshake: - The Charge Point must use TLS version 1.2 or above At least the following set of cipher suites must be supported: (TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 AND TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384) OR (TLS_RSA_WITH_AES_128_GCM_SHA256 AND TLS_RSA_WITH_AES_256_GCM_SHA384) - When using RSA or DSA the key must be at least 2048 bits long. and when using elliptic curve cryptography the key must be at least 224 bits long. - The received Client side certificate must be transmitted in the X.509 format encoded in Privacy-Enhanced Mail (PEM) format. - The certificate must include a serial number. - The subject field of the certificate must contain a commonName RDN which consists of the unique serial number of the Charge Point. *NOTE: If one of the above validations fails, the OCTT can still setup the WebSocket connection (if it is able to), but the testcase will FAIL and the OCTT reports why it failed.*	
	Post scenario validations: N/a	

2.25. Reusable states

Table 103. Reusable state: GetConfiguration

State	**GetConfiguration**	
Description	This state will retrieve a single configuration item from the Charge Point.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetConfiguration.conf**	**1.** The Central Systems sends a **GetConfiguration.req**
Tool validation(s)	n/a	
Expected result(s) / behaviour	**State** is *GetConfiguration*	

Table 104. Reusable state: Authorized

State	**Authorized**	
Description	This state will prepare the Charge Point.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	Manual Action: *Present idTag <Configured Valid IdTag>*	
	[Step 1 and step 3 may be reversed] **1.** The Charge Point sends an **Authorize.req**	**2.** The Central System responds with an **Authorize.conf** - **idTagInfo.status** is *Accepted*
	[Only expected if the status was not already Preparing] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **Authorize.req**) - **idTag** should be *<Configured Valid IdTag>* * Step 3: (Message: **StatusNotification.req**) - **status** should be *Preparing*	
Expected result(s) / behaviour	**State** is *Authorized*	

Table 105. Reusable state: Charging

State	**Charging**
Description	This state will start a transaction on the Charge Point using plug-in first and a remote start.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

State	**Charging**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetConfiguration.conf**	**1.** The Central System sends a **GetConfiguration.req** - **key[0]** is *AuthorizeRemoteTxRequests*
	Manual Action: *Plugin cable on both EV and CS side*	
	[If **status** is not *Reserved*] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	6. The Charge Point responds with a **RemoteStartTransaction.conf**	**5.** The Central System sends a **RemoteStartTransaction.req** - **connectorId** is <Configured ConnectorId> - **idTag** is *<Configured Valid IdTag>*
	[If **status** was *Reserved*] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
	[If **AuthorizeRemoteTxRequests** is *true*] **9.** The Charge Point sends an **Authorize.req**	**10.** The Central System responds with an **Authorize.conf** - **idTagInfo.status** is *Accepted*
	[Steps 11 and 13 may be reversed] **11.** The Charge Point sends a **StartTransaction.req**	**12.** The Central System responds with a **StartTransaction.conf** - **idTagInfo.status** is *Accepted*
	13. The Charge Point sends a **StatusNotification.req**	**14.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) (only if the **status** is not *Reserved*) - **connectorId** should be *<Configured ConnectorId>* - **status** should be *Preparing* * Step 6: (Message: **RemoteStartTransaction.conf**) - **status** should be *Accepted* * Step 8: (Message: **StatusNotification.req**) (only if the **status** was *Reserved*) - **connectorId** should be *<Configured ConnectorId>* - **status** should be *Preparing* * Step 9: (Message: **Authorize.req**) - **idTag** should be *<Configured Valid IdTag>* * Step 11: (Message: **StartTransaction.req**) - **connectorId** should be *<Configured ConnectorId>* - **idTag** should be *<Configured Valid IdTag>* * Step 13: (Message: **StatusNotification.req**) - **connectorId** should be *<Configured ConnectorId>* - **status** should be *Charging*	
Expected result(s) / behaviour	**State** is *Charging*	

Table 106. Reusable state: SetConnectorFaulted

State	**SetConnectorFaulted**	
Description	This state will set a single connector of the Charge Point to Unavailable.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	Manual Action: *Put the connector into a Faulted state.*	
	1. The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) - **status** should be *Faulted* - **connectorId** should be *<Configured ConnectorId>*	
Expected result(s) / behaviour	**State** is *SetConnectorFaulted*	

Table 107. Reusable state: SetChargePointFaulted

State	**SetChargePointFaulted**	
Description	This state will set the whole Charge Point to Unavailable.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	Manual Action: *Put the Charge Point into a Faulted state.*	
	1. The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	Note: Steps 3 and 4 will be repeated for every connector and connector = 0.	
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) - **status** should be *Faulted*	
Expected result(s) / behaviour	**State** is *SetChargePointFaulted*	

Table 108. Reusable state: SetConnectorUnavailable

State	**SetConnectorUnavailable**
Description	This state will set a single connector of the Charge Point to Unavailable.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

State	**SetConnectorUnavailable**	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeAvailability.conf**	**1.** The Central System sends a **ChangeAvailability.req** - **type** is *Inoperative* - **connectorId** is *<Configured ConnectorId>*
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **ChangeAvailability.conf**) - **status** should be *Accepted* * Step 3: (Message: **StatusNotification.req**) - **status** should be *Unavailable* - **connectorId** should be *<Configured ConnectorId>*	
Expected result(s) / behaviour	**State** is *SetConnectorUnavailable*	

Table 109. Reusable state: SetChargePointUnavailable

State	**SetChargePointUnavailable**	
Description	This state will set the whole Charge Point to Unavailable.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ChangeAvailability.conf**	**1.** The Central System sends a **ChangeAvailability.req** - **type** is *Inoperative* - **connectorId** is *0*
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	Note: Steps 3 and 4 will be repeated for every connector and connector = 0.	
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Accepted* * Step 3: (Message: **StatusNotification.req**) - **status** should be *Unavailable*	
Expected result(s) / behaviour	**State** is *SetChargePointUnavailable*	

Table 110. Reusable state: SetConnectorOccupied

State	**SetConnectorOccupied**
Description	This state will occupy a single connector of the Charge Point.

State	**SetConnectorOccupied**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	[EV driver plugs in cable] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) - **status** should be *Preparing* - **connectorId** should be *<Configured ConnectorId>*	
Expected result(s) / behaviour	**State** is *SetConnectorOccupied*	

Table 111. Reusable state: Reserved

State	**Reserved**	
Description	This state will reserve a connector on the Charge Point.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req** - **connectorId** is *<Configured ConnectorId>* - **idTag** is *<Configured Valid IdTag>*
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - **status** should be *Accepted* * Step 3: (Message: **StatusNotification.req**) - **status** should be *Reserved* - **connectorId** should be *<Configured ConnectorId>*	
Expected result(s) / behaviour	**State** is *Reserved*	

2.26. Memory states

Table 112. Memory state: IdTagCached

State	**IdTagCached**
Description	This state will ensure that an idTag is cached at the Charge Point.

State	**IdTagCached**	
Before	**Configuration State(s):** - **AuthorizationCacheEnabled** is *true*	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	Manual Action: *Present idTag <Configured Valid IdTag>*	
	[Steps 1 and 3 may be reversed] **1.** The Charge Point sends a **StopTransaction.req**	**2.** The Central System responds with a **StopTransaction.conf** - **idTagInfo.status** is *Accepted*
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	Manual Action: *Unplug cable on both EV and CS side*	
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StopTransaction.req**) - **transactionId** should be *<transactionId generated at Charging>* * Step 3: (Message: **StatusNotification.req**) - **connectorId** should be *<Configured ConnectorId>* - **status** should be *Finishing* * Step 5: (Message: **StatusNotification.req**) - **connectorId** should be *<Configured ConnectorId>* - **status** should be *Available*	
Expected result(s) / behaviour	**State** is *IdTagCached*	

Table 113. Memory state: IdTagLocalAuthList

State	**IdTagLocalAuthList**	
Description	This state will ensure that an idTag is in the local authorization list of the Charge Point.	
Before	**Configuration State(s):** - **LocalAuthListEnabled** is *true*	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SendLocalList.conf**	**1.** The Central System sends a **SendLocalList.req** - **listVersion** is *1* Field **localAuthorizationList[0]**: - **idTag** is *<Configured Valid IdTag>* - **idTagInfo.status** is *Accepted*
Tool validation(s)	* Step 2: (Message: **SendLocalList.conf**) - **status** should be *Accepted*	
Expected result(s) / behaviour	**State** is *IdTagLocalAuthList*	

Table 114. Memory state: CertificateInstalled

State	**CertificateInstalled**	
Description	This state will ensure that a root certificate is installed on the Charge Point.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**1.** The Central System sends a **GetInstalledCertificateIds.req**
	4. The Charge Point responds with a **InstallCertificate.conf**	[Only send if the certificate is not already installed] **3.** The Central System sends a **InstallCertificate.req**
Tool validation(s)	* Step 2: (Message: **GetInstalledCertificateIds.conf**) - **status** should be *Accepted* * Step 4: (Message: **InstallCertificate.conf**) - **status** should be *Accepted*	
Expected result(s) / behaviour	**State** is *CertificateInstalled*	

Table 115. Memory state: RenewChargePointCertificate

State	**RenewChargePointCertificate**	
Description	This state will ensure that a client certificate is installed on the Charge Point.	
Before	**Configuration State(s):** - **CpoName** is *<Configured Vendor Name>*	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **ExtendedTriggerMessage.conf**	**1.** The Central System sends a **ExtendedTriggerMessage.req** - **requestedMessage** is *SignChargePointCertificate* - **connectorId** is *<Omitted>*
	[The Charge Point generates a new public/private key pair and generates a Certificate Signing Request.] **3.** The Charge Point sends a **SignCertificate.req**	**4.** The Central System responds with a **SignCertificate.conf** - **status** is *Accepted*
	[The Charge Point verifies the validity of the signed certificate.] **6.** The Charge Point responds with a **CertificateSigned.conf**	[Certificate Authority Server signs the certificate.] **5.** The Central System sends a **CertificateSigned.req**
Tool validation(s)	* Step 2: (Message: **ExtendedTriggerMessage.conf**) - **status** should be *Accepted* * Step 6: (Message: **CertificateSigned.conf**) - **status** should be *Accepted*	

State	**RenewChargePointCertificate**
Expected result(s) / behaviour	**State** is *RenewChargePointCertificate*

Table 116. Memory state: SetChargingProfile

State	**SetChargingProfile**	
Description	This state will set a *ChargingProfile* on the Charge Point.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (SUT)**	**Central System (Tool)**
	2. The Charge Point responds with a **SetChargingProfile.conf**	**1.** The Central System sends a **SetChargingProfile.req** - **connectorId** is *<Configured ConnectorId>*
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) - **status** should be *Accepted*	
Expected result(s) / behaviour	**State** is *SetChargingProfile*	

3. System Under Test (SUT) Central System

This section contains all test cases available in the tool, when configured System Under Test (SUT) Central System.

3.1. Cold Boot Charge Point

3.1.1. Cold Boot Charge Point

Table 117. Test Case Id: TC_001_CSMS

Test case name	**Cold Boot Charge Point**	
Test case Id	TC_001_CSMS	
Description	This scenario is used to startup the Charge Point and let it register itself at the Central System.	
Purpose	To test if the Central System is able to handle a boot process.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends a **BootNotification.req**	**2.** The Central System responds with a **BootNotification.conf**
	[Send a StatusNotification per connector and connectorId=0.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	[Every x seconds.] **5.** The Charge Point sends a **Heartbeat.req**	**6.** The Central System responds with a **Heartbeat.conf**
Tool validation(s)	* Step 1: (Message: **BootNotification.req**) * Step 3: (Message: **StatusNotification.req**) **status** is *Available* * Step 5: (Message: **Heartbeat.req**) *Send a Heartbeat.req every x seconds. x equals **interval** from step 2.*	* Step 2: (Message: **BootNotification.conf**) The **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.2. Start Charging Session

3.2.1. Regular Charging Session - Plugin First

Table 118. Test Case Id: TC_003_CSMS

Test case name	**Regular Charging Session - Plugin First**
Test case Id	TC_003_CSMS
Description	This scenario is used to start a Charging session.
Purpose	To test if the Central System can handle when the Charge Point starts a Charging Session when first doing plugin cable.
Prerequisite(s)	n/a

Test case name	**Regular Charging Session - Plugin First**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver plugs in the cable.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	[EV driver presents identification.] **3.** The Charge Point sends an **Authorize.req**	**4.** The Central System responds with an **Authorize.conf**
	5. The Charge Point sends a **StartTransaction.req**	**6.** The Central System responds with a **StartTransaction.conf**
	7. The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 7: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 4: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted* * Step 6: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.2.2. Regular Charging Session – Identification First

Table 119. Test Case Id: TC_004_1_CSMS

Test case name	**Regular Charging Session – Identification First**	
Test case Id	TC_004_1_CSMS	
Description	This scenario is used to start a charging session.	
Purpose	To test if the Central System can handle when the Charge Point starts a charging session when first doing authorization.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	- *Charging*	
Tool validation(s)	n/a	n/a
Expected result(s) / behaviour	n/a	n/a

3.2.3. Regular Charging Session – Identification First - ConnectionTimeOut

Table 120. Test Case Id: TC_004_2_CSMS

Test case name	**Regular Charging Session – Identification First - ConnectionTimeOut**
Test case Id	TC_004_2_CSMS
Description	This scenario is used to make a connector available when it is not used.

Test case name	**Regular Charging Session – Identification First - ConnectionTimeOut**	
Purpose	To test if the Central System can handle when the Charge Point sets the connector back to *Available*, when the connectionTimeOut is reached.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Authorized*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	[After the configured connectionTimeOut has expired.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 3: (Message: **StatusNotification.req**) **status** is *Available*	n/a
Expected result(s) / behaviour	n/a	n/a

3.2.4. EV Side Disconnected - StopTransactionOnEVSideDisconnect = true - UnlockConnectorOnEVSideDisconnect = true

Table 121. Test Case Id: TC_005_1_CSMS

Test case name	**EV Side Disconnected - StopTransactionOnEVSideDisconnect = true - UnlockConnectorOnEVSideDisconnect = true**	
Test case Id	TC_005_1_CSMS	
Description	This scenario is used to stop the transaction when the cable is disconnected at EV side.	
Purpose	To test if the Central System can handle when the Charge Point stops the transaction when the cable is disconnected at EV side, and it is configured to do so.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver disconnects cable on EV side.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	3. The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[EV driver unplugs the cable from the Charge Point.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**

Test case name	**EV Side Disconnected - StopTransactionOnEVSideDisconnect = true - UnlockConnectorOnEVSideDisconnect = true**	
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *SuspendedEV* * Step 3: (Message: **StopTransaction.req**) **reason** is *EVDisconnected* * Step 5: (Message: **StatusNotification.req**) **status** is *Finishing* * Step 7: (Message: **StatusNotification.req**) **status** is *Available*	n/a
Expected result(s) / behaviour	n/a	n/a

3.3. Cache

3.3.1. Regular Start Charging Session – Cached Id

Table 122. Test Case Id: TC_007_CSMS

Test case name	**Regular Start Charging Session – Cached Id**	
Test case Id	TC_007_CSMS	
Description	This scenario is used to start a transaction with an id stored in the Authorization cache.	
Purpose	To test if the Central System is able to handle a Charge Point starting a transaction with an id which is stored in the Authorization cache.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver plugs in the cable.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	[EV driver presents identification.] **3.** The Charge Point sends a **StartTransaction.req**	**4.** The Central System responds with a **StartTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 5: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 4: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.3.2. Clear Authorization Data in Authorization Cache

Table 123. Test Case Id: TC_061_CSMS

Test case name	**Clear Authorization Data in Authorization Cache**	
Test case Id	TC_061_CSMS	
Description	The Central System can clear the Authorization Cache of a Charge Point.	
Purpose	Check whether the Central System can clear the Authorization Cache of a Charge Point.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ClearCache.conf**	**1.** The Central System sends a **ClearCache.req**
Tool validation(s)	* Step 2: (Message: **ClearCache.conf**) **status** is *Accepted*	n/a
Expected result(s) / behaviour	The Charge Point Authorization Cache is cleared.	The Central System is able to send a message to clear the cache.

3.4. Core Profile - Remote actions Happy flow

3.4.1. Remote Start Charging Session – Cable Plugged in First

Table 124. Test Case Id: TC_010_CSMS

Test case name	**Remote Start Charging Session – Cable Plugged in First**	
Test case Id	TC_010_CSMS	
Description	This scenario is used to start a transaction remotely.	
Purpose	To test if the Central System can handle when a Charge point starts a transaction after receiving a RemoteStartTransaction.req from the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver plugs in the cable.] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	4. The Charge Point responds with a **RemoteStartTransaction.conf**	**3.** The Central System sends a **RemoteStartTransaction.req**
	5. The Charge Point sends an **Authorize.req**	**6.** The Central System responds with an **Authorize.conf**
	7. The Charge Point sends a **StartTransaction.req**	**8.** The Central System responds with a **StartTransaction.conf**
	9. The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 4: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 9: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 6: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted* * Step 8: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.4.2. Remote Start Charging Session – Remote Start First

Table 125. Test Case Id: TC_011_1_CSMS

Test case name	**Remote Start Charging Session – Remote Start First**
Test case Id	TC_011_1_CSMS
Description	This scenario is used to start a transaction remotely.
Purpose	To test if the Central System can handle when a Charge point starts a transaction after receiving a RemoteStartTransaction.req from the Central System.
Prerequisite(s)	n/a

Test case name	**Remote Start Charging Session – Remote Start First**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **RemoteStartTransaction.conf**	**1.** The Central System sends a **RemoteStartTransaction.req**
	3. The Charge Point sends an **Authorize.req**	**4.** The Central System responds with an **Authorize.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[EV driver plugs in the cable.] **7.** The Charge Point sends a **StartTransaction.req**	**8.** The Central System responds with a **StartTransaction.conf**
	9. The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 5: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 9: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 6: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted* * Step 8: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.4.3. Remote Start Charging Session – Time Out

Table 126. Test Case Id: TC_011_2_CSMS

Test case name	**Remote Start Charging Session – Time Out**
Test case Id	TC_011_2_CSMS
Description	This scenario is used to set a connector back to available, after receiving a RemoteStartTransaction.req and it takes to long to plugin the cable.
Purpose	To test if the Central System can handle when a Charge Point sets the connector back to available, after reaching the configured connection timeout.
Prerequisite(s)	n/a
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Remote Start Charging Session – Time Out**	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **RemoteStartTransaction.conf**	**1.** The Central System sends a **RemoteStartTransaction.req**
	3. The Charge Point sends an **Authorize.req**	**4.** The Central System responds with an **Authorize.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[After the configured connection timeout has been reached.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **RemoteStartTransaction.conf**) **status** is *Accepted* * Step 5: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 7: (Message: **StatusNotification.req**) **status** is *Available*	* Step 4: (Message: **Authorize.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.4.4. Remote Stop Charging Session

Table 127. Test Case Id: TC_012_CSMS

Test case name	**Remote Stop Charging Session**	
Test case Id	TC_012_CSMS	
Description	This scenario is used to remotely stop a transaction.	
Purpose	To test if the Central System can remotely stop a transaction.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **RemoteStopTransaction.conf**	**1.** The Central System sends a **RemoteStopTransaction.req**
	3. The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	[EV driver unplugs the cable.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**

Test case name	**Remote Stop Charging Session**	
Tool validation(s)	* Step 2: (Message: **RemoteStopTransaction.conf**) **status** is *Accepted* * Step 3: (Message: **StopTransaction.req**) **reason** is *Remote* * Step 5: (Message: **StatusNotification.req**) **status** is *Finishing* * Step 7: (Message: **StatusNotification.req**) **status** is *Available*	n/a
Expected result(s) / behaviour	n/a	n/a

3.5. Core Profile - Resetting Happy Flow

3.5.1. Hard Reset

Table 128. Test Case Id: TC_013_CSMS

Test case name	**Hard Reset**	
Test case Id	TC_013_CSMS	
Description	This scenario is used to hard reset a Charge Point.	
Purpose	To test if the Central System is able to trigger a hard reset.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **Reset.conf**	**1.** The Central System sends a **Reset.req**
	3. The Charge Point sends a **BootNotification.req**	**4.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **Reset.conf**) **status** is *Accepted* * Step 5: (Message: **StatusNotification.req**) **status** is *Available*	* Step 1: (Message: **Reset.req**) The **type** is *Hard* * Step 4: (Message: **BootNotification.conf**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.5.2. Soft Reset

Table 129. Test Case Id: TC_014_CSMS

Test case name	**Soft Reset**	
Test case Id	TC_014_CSMS	
Description	This scenario is used to soft reset a Charge Point.	
Purpose	To test if the Central System is able to trigger a soft reset.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **Reset.conf**	**1.** The Central System sends a **Reset.req**
	3. The Charge Point sends a **BootNotification.req**	**4.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**

Test case name	**Soft Reset**	
Tool validation(s)	* Step 2: (Message: **Reset.conf**) **status** is *Accepted* * Step 5: (Message: **StatusNotification.req**) **status** is *Available*	* Step 1: (Message: **Reset.req**) The **type** is *Soft* * Step 4: (Message: **BootNotification.conf**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.6. Core Profile - Unlocking Happy flow

3.6.1. Unlock connector - no charging session running (Not fixed cable)

Table 130. Test Case Id: TC_017_1_CSMS

Test case name	**Unlock connector - no charging session running (Not fixed cable)**	
Test case Id	TC_017_1_CSMS	
Description	This scenario is used to unlock a connector of a Charge Point.	
Purpose	To test if the Central System can handle when the Charge Point unlocks the connector, when requested by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *Unlocked*	n/a
Expected result(s) / behaviour	n/a	n/a

3.6.2. Unlock connector - no charging session running (Fixed cable)

Table 131. Test Case Id: TC_017_2_CSMS

Test case name	**Unlock connector - no charging session running (Fixed cable)**	
Test case Id	TC_017_2_CSMS	
Description	This scenario describes how to Charge Point should react to an UnlockConnector.req, when having a fixed cable.	
Purpose	To test if the Central System can handle when the Charge Point notifies the Central System that it does not support the unlocking of a connector.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *NotSupported*	n/a
Expected result(s) / behaviour	n/a	n/a

3.6.3. Unlock Connector - With Charging Session

Table 132. Test Case Id: TC_018_1_CSMS

Test case name	**Unlock Connector - With Charging Session (Not fixed cable)**	
Test case Id	TC_018_1_CSMS	
Description	This scenario is used to unlock a connector of a Charge Point, while a transaction is ongoing.	
Purpose	To test if the Central System can handle when the Charge Point unlocks the connector, when requested by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5 The Charge Point sends a **StopTransaction.req**	**6.** The Central System responds with a **StopTransaction.conf**
	[EV driver unplugs the cable.] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *Unlocked* * Step 3: (Message: **StatusNotification.req**) **status** is *Finishing* * Step 5: (Message: **StopTransaction.req**) **reason** is *UnlockCommand* * Step 7: (Message: **StatusNotification.req**) **status** is *Available*	n/a
Expected result(s) / behaviour	n/a	n/a

3.7. Core Profile - Configuration Happy flow

3.7.1. Retrieve all configuration keys

Table 133. Test Case Id: TC_019_1_CSMS

Test case name	**Retrieve all configuration keys**	
Test case Id	TC_019_1_CSMS	
Description	The Central System is able to retrieve all available configuration keys.	
Purpose	To check whether the Central System is able to retrieve all Configuration keys and whether the Charge Point has all required keys configured.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetConfiguration.conf**.	**1.** The Central Systems sends a **GetConfiguration.req** message to the Charge Point.

Test case name	**Retrieve all configuration keys**	
Tool validation(s)	* Step 2: (Message: **GetConfiguration.conf**) - **accessibility** contains the following values. **Core:** **Configuration Key / accessibility** AuthorizeRemoteTxRequests / R or RW ClockAlignedDataInterval / RW ConnectionTimeOut / RW ConnectorPhaseRotation / RW GetConfigurationMaxKeys / R HeartbeatInterval / RW LocalAuthorizeOffline / RW LocalPreAuthorize / RW MeterValuesAlignedData / RW MeterValuesSampledData / RW MeterValueSampleInterval / RW NumberOfConnectors / R ResetRetries / RW StopTransactionOnEVSideDisconnect / RW StopTransactionOnInvalidId / RW StopTxnAlignedData / RW StopTxnSampledData / RW SupportedFeatureProfiles / R TransactionMessageAttempts / RW TransactionMessageRetryInterval / RW UnlockConnectorOnEVSideDisconnect / RW **Local Auth List Management:** LocalAuthListEnabled / RW LocalAuthListMaxLength / R SendLocalListMaxLength / R **Smart Charging Profile:** ChargeProfileMaxStackLevel / R ChargingScheduleAllowedChargingRateUnit / R ChargingScheduleMaxPeriods / R MaxChargingProfilesInstalled / R **Reservation:** *None* **Remote Trigger:** *None*	* Step 1: (Message: **GetConfiguration.req**) The **key** is *<Empty>*
Expected result(s) / behaviour	All required keys are configured.	The Central System is able to retrieve the values of all requested configuration keys.

3.7.2. Retrieve specific configuration key

Table 134. Test Case Id: TC_019_2_CSMS

Test case name	**Retrieve specific configuration key**
Test case Id	TC_019_2_CSMS
Description	The Central System is able to retrieve a specific configuration key.
Purpose	To check whether the Central System is able to retrieve a specific Configuration key.
Prerequisite(s)	n/a

Test case name	**Retrieve specific configuration key**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**System Under Test: Central System**
	2. The Charge Point responds with a **GetConfiguration.conf**.	**1.** The Central Systems sends a **GetConfiguration.req** message to the Charge Point.
Tool validation(s)	* Step 2: (Message: **GetConfiguration.conf**) **unknownKey** list is *<Empty>* **configurationKey.key** should be *SupportedFeatureProfiles*	* Step 1: (Message: **GetConfiguration.req**) The **key** is *SupportedFeatureProfiles*
Expected result(s) / behaviour	n/a	The Central System is able to retrieve the value of the requested configuration key.

3.7.3. Change/set Configuration

Table 135. Test Case Id: TC_021_CSMS

Test case name	**Change/set Configuration**	
Test case Id	TC_021_CSMS	
Description	This scenario is used to set the value of a configuration key.	
Purpose	To test if the Central System can handle when a Charge Point sets the configuration key value, specified by the Central System.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) **status** is *Accepted*	* Step 1: (Message: **ChangeConfiguration.req**) The **key** is *MeterValueSampleInterval* The **value** is *60*
Expected result(s) / behaviour	n/a	n/a

3.8. Core Profile - Basic Actions Non-happy flow

3.8.1. Start Charging Session – Authorize invalid

Table 136. Test Case Id: TC_023_1_CSMS

Test case name	**Start Charging Session – Authorize invalid**	
Test case Id	TC_023_1_CSMS	
Description	This scenario is used to inform the Charge Point that the EV Driver is not Authorized to start a transaction.	
Purpose	To test if the Central System is able to provide an invalid response on an Authorize.req.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver presents invalid identification.] **1.** The Charge Point sends an **Authorize.req**	**2.** The Central System responds with an **Authorize.conf**
Tool validation(s)	n/a	* Step 1: (Message: **Authorize.conf**) **idTagInfo.status** is *Invalid*
Expected result(s) / behaviour	n/a	n/a

3.8.2. Start Charging Session – Authorize expired

Table 137. Test Case Id: TC_023_2_CSMS

Test case name	**Start Charging Session – Authorize expired**	
Test case Id	TC_023_2_CSMS	
Description	This scenario is used to inform the Charge Point that the EV Driver is not Authorized to start a transaction.	
Purpose	To test if the Central System is able to provide an expired response on an Authorize.req.	
Prerequisite(s)	The Central System has an idTag in memory with status 'Expired'.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver presents expired identification.] **1.** The Charge Point sends an **Authorize.req**	**2.** The Central System responds with an **Authorize.conf**
Tool validation(s)	n/a	* Step 1: (Message: **Authorize.conf**) **idTagInfo.status** is *Expired*
Expected result(s) / behaviour	n/a	n/a

3.8.3. Start Charging Session – Authorize blocked

Table 138. Test Case Id: TC_023_3_CSMS

Test case name	**Start Charging Session – Authorize blocked**	
Test case Id	TC_023_3_CSMS	
Description	This scenario is used to inform the Charge Point that the EV Driver is not Authorized to start a transaction.	
Purpose	To test if the Central System is able to provide a blocked response on an Authorize.req.	
Prerequisite(s)	- The Central System has an idTag in memory with status 'Blocked'.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver presents blocked identification.] **1.** The Charge Point sends an **Authorize.req**	**2.** The Central System responds with an **Authorize.conf**
Tool validation(s)	n/a	* Step 1: (Message: **Authorize.conf**) **idTagInfo.status** is *Blocked*
Expected result(s) / behaviour	n/a	n/a

3.8.4. Start Charging Session Lock Failure

Table 139. Test Case Id: TC_024_CSMS

Test case name	**Start Charging Session Lock Failure**	
Test case Id	TC_024_CSMS	
Description	This scenario is used to report a connector lock failure.	
Purpose	To test if the Central System is able to handle a report of a connector lock failure.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Authorized*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	[EV driver plugs in the cable.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) **status** is *Preparing* * Step 3: (Message: **StatusNotification.req**) **errorCode** is *ConnectorLockFailure* **status** is *Faulted*	n/a
Expected result(s) / behaviour	n/a	n/a

3.9. Core Profile - Remote Actions Non-Happy Flow

3.9.1. Remote Start Charging Session – Rejected

Table 140. Test Case Id: TC_026_CSMS

Test case name	**Remote Start Charging Session – Rejected**	
Test case Id	TC_026_CSMS	
Description	This scenario is used to reject a RemoteStartTransaction.req.	
Purpose	To test if the Central System can handle when a Charge Point rejects a RemoteStartTransaction.req.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **RemoteStartTransaction.conf**	[The CPO remotely requests a start transaction.] **1.** The Central System sends a **RemoteStartTransaction.req**
Tool validation(s)	* Step 2: (Message: **RemoteStartTransaction.conf**) **status** is *Rejected*	n/a
Expected result(s) / behaviour	n/a	n/a

3.9.2. Remote Stop Transaction – Rejected

Table 141. Test Case Id: TC_028_CSMS

Test case name	**Remote Stop Transaction – Rejected**	
Test case Id	TC_028_CSMS	
Description	This scenario is used to reject a RemoteStopTransaction.req, when an unknown transactionId is given.	
Purpose	To test if the Central System can handle when a Charge Point rejects a RemoteStopTransaction.req, when an unknown transactionId is given.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **RemoteStopTransaction.conf**	**1.** The Central System sends a **RemoteStopTransaction.req**
Tool validation(s)	* Step 2: (Message: **RemoteStopTransaction.conf**) **status** is *Rejected*	n/a
Expected result(s) / behaviour	n/a	n/a

3.10. Core Profile - Unlocking Non-happy flow

3.10.1. Unlock Connector – Unlock Failure

Table 142. Test Case Id: TC_030_CSMS

Test case name	**Unlock Connector – Unlock Failure**	
Test case Id	TC_030_CSMS	
Description	This scenario is used to report a connector lock failure.	
Purpose	To test if the Central System is able to handle a report of a connector lock failure.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *UnlockFailed*	n/a
Expected result(s) / behaviour	n/a	n/a

3.10.2. Unlock Connector – Unknown Connector

Table 143. Test Case Id: TC_031_CSMS

Test case name	**Unlock Connector – Unknown Connector**	
Test case Id	TC_031_CSMS	
Description	This scenario is used to reject an UnlockConnector.req, when an unknown connectorId is given.	
Purpose	To test if the Central System is able to handle a Charge Point that does not support UnlockConnector.req.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UnlockConnector.conf**	**1.** The Central System sends a **UnlockConnector.req**
Tool validation(s)	* Step 2: (Message: **UnlockConnector.conf**) **status** is *NotSupported*	n/a
Expected result(s) / behaviour	n/a	n/a

3.11. Core Profile - Power Failure Non-Happy Flow

3.11.1. Power failure boot charging point-configured to stop transaction(s)

Table 144. Test Case Id: TC_032_1_CSMS

Test case name	**Power failure boot charging point-configured to stop transaction(s)**	
Test case Id	TC_032_1_CSMS	
Description	This scenario is used to stop all transactions, when a power failure occurred.	
Purpose	To test if the Central System can handle when a Charge Point stops all transactions, when a power failure occurred.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[Disconnect and reconnect the power of the Charge Point.] **1.** The Charge Point sends a **BootNotification.req**	**2.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId = 0.] **3.** The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5. The Charge Point sends a **StopTransaction.req**	**6.** The Central System responds with a **StopTransaction.conf**
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) **connectorId** is *<The connector which had the ongoing transaction>* **status** is *Finishing* (Message: **StatusNotification.req**) *The other StatusNotification messages.* **status** is *Available* * Step 5: (Message: **StopTransaction.req**) **reason** is *PowerLoss*	* Step 2: (Message: **BootNotification.req**) **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.12. Core Profile - Offline behavior Non-Happy Flow

3.12.1. Offline Start Transaction - Valid IdTag

Table 145. Test Case Id: TC_037_1_CSMS

Test case name	**Offline Start Transaction - Valid IdTag**	
Test case Id	TC_037_1_CSMS	
Description	This scenario is used to start a transaction, while being offline.	
Purpose	To test if the Central System can handle when a Charge Point starts a transaction, while being offline and queues transaction-related messages, after restoring the connection.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[Remove connectivity between Charge Point and Central System.] [EV Driver starts offline a transaction with a valid idTag.] [Restore connectivity between Charge Point and Central System.] **1.** The Charge Point sends a **StartTransaction.req**	**2.** The Central System responds with a **StartTransaction.conf**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) **status** is *Charging*	* Step 2: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.12.2. Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = true

Table 146. Test Case Id: TC_037_3_CSMS

Test case name	**Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = true**
Test case Id	TC_037_3_CSMS
Description	This scenario is used to start a transaction, while being offline.
Purpose	To test if the Central System can handle when a Charge Point starts a transaction, while being offline and queues transaction-related messages, after restoring the connection.
Prerequisite(s)	n/a
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Offline Start Transaction - Invalid IdTag - StopTransactionOnInvalidId = true**	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[Remove connectivity between Charge Point and Central System.] [EV Driver starts offline a transaction with an invalid idTag.] [Restore connectivity between Charge Point and Central System.] **1.** The Charge Point sends a **StartTransaction.req**	**2.** The Central System responds with a **StartTransaction.conf**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5. The Charge Point sends a **StopTransaction.req**	**6.** The Central System responds with a **StopTransaction.conf**
	7. The Charge Point sends a **StatusNotification.req**	**8.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) **status** is *Charging* * Step 5: (Message: **StopTransaction.req**) **reason** is *DeAuthorized* * Step 7 (Message: **StatusNotification.req**) **status** is *Finishing*	* Step 2: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Invalid*
Expected result(s) / behaviour	n/a	n/a

3.12.3. Offline Transaction

Table 147. Test Case Id: TC_039_CSMS

Test case name	**Offline Transaction**	
Test case Id	TC_039_CSMS	
Description	This scenario is used to start and stop a transaction, while the Charge Point is offline.	
Purpose	To test if the Central System is able to handle queued transaction-related messages, after a Charge Point comes back online again.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[Remove connectivity between Charge Point and Central System.] [EV Driver starts offline a transaction.] [EV Driver stops offline a transaction.] [EV driver unplugs the cable.] [Restore connectivity between Charge Point and Central System.] **1.** The Charge Point sends a **StartTransaction.req**	**2.** The Central System responds with a **StartTransaction.conf**
	3. The Charge Point sends a **StopTransaction.req**	**4.** The Central System responds with a **StopTransaction.conf**

Test case name	**Offline Transaction**	
Tool validation(s)	* Step 3: (Message: **StopTransaction.req**) **reason** is *Local*	* Step 2: (Message: **StartTransaction.conf**) **idTagInfo.status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

3.13. Core Profile - Configuration Keys Non-Happy Flow

3.13.1. Configuration keys

Table 148. Test Case Id: TC_040_1_CSMS

Test case name	**Configuration keys**	
Test case Id	TC_040_1_CSMS	
Description	This scenario is used to reject an unknown configuration key.	
Purpose	To test if the Central System is able to handle a Charge Point that does not support a given configuration key.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) The **status** is *NotSupported*	n/a
Expected result(s) / behaviour	n/a	n/a

3.13.2. Configuration Keys

Table 149. Test Case Id: TC_040_2_CSMS

Test case name	**Configuration keys**	
Test case Id	TC_040_2_CSMS	
Description	This scenario is used to reject setting a configuration key, when an incorrect value is given.	
Purpose	To test if the Central System is able to handle a Charge Point rejecting setting a configuration key.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) The **status** is *Rejected*	* Step 1: (Message: **ChangeConfiguration.req**) The **key** is *MeterValueSampleInterval* **value** is *-1*
Expected result(s) / behaviour	n/a	n/a

3.14. Local Authorization List

3.14.1. Get Local List Version

Get Local List Version (not supported)

Table 150. Test Case Id: TC_042_1_CSMS

Test case name	**Get Local List Version (not supported)**	
Test case Id	TC_042_1_CSMS	
Description	The Central System can request a Charge Point for the version number of the Local Authorization List.	
Purpose	Check whether a Central System is able to retrieve the local list version from a Charge Point.	
Prerequisite(s)	The Central System supports the Local Auth List Management feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetLocalListVersion.conf**.	**1.** The Central System sends a **GetLocalListVersion.req**.
Tool validation(s)	* Step 2: (Message: **GetLocalListVersion.conf**) **listVersion** is *-1*	n/a
Expected result(s) / behaviour	n/a	n/a

Get Local List Version (empty)

Table 151. Test Case Id: TC_042_2_CSMS

Test case name	**Get Local List Version (empty)**	
Test case Id	TC_042_2_CSMS	
Description	The Central System can request a Charge Point for the version number of the Local Authorization List.	
Purpose	Check whether a Central System is able to retrieve the local list version from a Charge Point.	
Prerequisite(s)	The Central System supports the Local Auth List Management feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetLocalListVersion.conf**.	**1.** The Central System sends a **GetLocalListVersion.req**.
Tool validation(s)	* Step 2: (Message: **GetLocalListVersion.conf**) **listVersion** is *0*	n/a
Expected result(s) / behaviour	n/a	n/a

3.14.2. Send Local Authorization List

Send Local Authorization List - NotSupported

Table 152. Test Case Id: TC_043_1_CSMS

Test case name	**Send Local Authorization List - NotSupported**	
Test case Id	TC_043_1_CSMS	
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.	
Purpose	To check whether a Central System can handle a *NotSupported* status, after sending a Local Authorization List.	
Prerequisite(s)	The Central System supports the Local Auth List Management feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **SendLocalList.conf**	**1.** The Central System sends a **SendLocalList.req**
Tool validation(s)	* Step 2: (Message: **SendLocalList**) - **Status** is *NotSupported*	* Step 1: (Message: **SendLocalList.req**) - **updateType** should be *Full*
Expected result(s) / behaviour	n/a	The Central System is able to send a local list and is able to receive a *NotSupported* response.

Send Local Authorization List - Failed

Table 153. Test Case Id: TC_043_3_CSMS

Test case name	**Send Local Authorization List - Failed**	
Test case Id	TC_043_3_CSMS	
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.	
Purpose	To check whether a Central System can handle a *Rejected* status, after sending a Local Authorization List.	
Prerequisite(s)	The Central System supports the Local Auth List Management feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **SendLocalList.conf**	**1.** The Central System sends a **SendLocalList.req**
Tool validation(s)	* Step 2: (Message: **SendLocalList**) - **Status** is *Failed*	* Step 1: (Message: **SendLocalList.req**) - **updateType** should be *Full*
Expected result(s) / behaviour	n/a	The Central System is able to send a local list and is able to receive a *Failed* response.

Send Local Authorization List - Full

Table 154. Test Case Id: TC_043_4_CSMS

Test case name	**Send Local Authorization List - Full**	
Test case Id	TC_043_4_CSMS	
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.	
Purpose	Check whether a Local Authorization List can be sent to a Charge Point to authorize an EV driver.	
Prerequisite(s)	The Central System supports the Local Auth List Management feature profile and has at least 1 IdToken to add to the local authorization list.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **SendLocalList.conf**	**1.** The Central System sends a **SendLocalList.req**
Tool validation(s)	* Step 2: (Message: **SendLocalList.conf**) - **Status** is *Accepted*	* Step 1: (Message: **SendLocalList.req**) - **UpdateType** should be *Full* - All **localAuthorizationList** entries have an **idTagInfo**
Expected result(s) / behaviour	n/a	The Central System is able to send a local list.

Send Local Authorization List - Differential

Table 155. Test Case Id: TC_043_5_CSMS

Test case name	**Send Local Authorization List - Differential**	
Test case Id	TC_043_5_CSMS	
Description	The Charge Point can authorize an EV driver based on a local list that is set by the Central System.	
Purpose	Check whether a Local Authorization List can be sent to a Charge Point to authorize an EV driver	
Prerequisite(s)	The Central System supports the Local Auth List Management feature profile and has at least 1 IdToken to add to the local authorization list.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetLocalListVersion.conf**	**1.** The Central System sends a **GetLocalListVersion.req**
	4. The Charge Point responds with a **SendLocalList.conf**	**3.** The Central System sends a **SendLocalList.req**
	Note: Messages 1 and 2 are optional.	
Tool validation(s)	* Step 4: (Message: **SendLocalList.conf**) - **Status** is *Accepted*	* Step 3: (Message: **SendLocalList.req**) - **updateType** should be *Differential* - All **localAuthorizationList** entries have an **idTagInfo**
Expected result(s) / behaviour	n/a	n/a

3.15. FirmwareManagement

3.15.1. Firmware Update - Download and Install

Table 156. Test Case Id: TC_044_1_CSMS

Test case name	**Firmware Update - Download and Install**	
Test case Id	TC_044_1_CSMS	
Description	The firmware of a Charge Point is updated.	
Purpose	Check whether Central System can trigger an update of the firmware of a Charge Point.	
Prerequisite(s)	The Central System supports the Firmware Management feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UpdateFirmware.conf**	**1.** The Central System sends a **UpdateFirmware.req**
	[The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **FirmwareStatusNotification.req**	**4.** The Central responds with a **FirmwareStatusNotification.conf**
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **FirmwareStatusNotification.req**	**6.** The Central responds with a **FirmwareStatusNotification.conf**
	[The Charge Point reports the status of all connectors] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central responds with a **StatusNotification.conf**
	[The Charge Point starts installing the firmware] **9.** The Charge Point sends a **FirmwareStatusNotification.req**	**10.** The Central responds with a **FirmwareStatusNotification.conf**
	11. The Charge Point sends a **BootNotification.req**	**12.** The Central responds with a **BootNotification.conf**
	[The Charge Point reports the status of all connectors] **13.** The Charge Point sends a **StatusNotification.req**	**14.** The Central responds with a **StatusNotification.conf**
	15. The Charge Point sends a **FirmwareStatusNotification.req**	**16.** The Central responds with a **FirmwareStatusNotification.conf**

Test case name	**Firmware Update - Download and Install**	
Tool validation(s)	* Step 3: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (Message: **StatusNotification.req**) The **status** is *Unavailable* * Step 9: (Message: **FirmwareStatusNotification.req**) The **status** is *Installing* * Step 13: (Message: **StatusNotification.req**) The **status** is *Available* * Step 15: (Message: **FirmwareStatusNotification.req**) The **status** is *Installed*	* Step 1: (Message: **UpdateFirmware.req**) The **firmware.location** is *<Firmware Download URL from test data>*
Expected result(s) / behaviour	n/a	n/a

3.15.2. Firmware Update - Download Failed

Table 157. Test Case Id: TC_044_2_CSMS

Test case name	**Firmware Update - Download Failed**	
Test case Id	TC_044_2_CSMS	
Description	The firmware of a Charge Point is being updated, but downloading the firmware fails.	
Purpose	Check whether Central System can handle messages for a firmware update in case downloading of the firmware fails.	
Prerequisite(s)	The Central System supports the Firmware Management feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UpdateFirmware.conf**	**1.** The Central System sends a **UpdateFirmware.req**
	[The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **FirmwareStatusNotification.req**	**4.** The Central responds with a **FirmwareStatusNotification.conf**
	[Downloading the firmware fails] **5.** The Charge Point sends a **FirmwareStatusNotification.req**	**6.** The Central responds with a **FirmwareStatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **FirmwareStatusNotification.req**) The **status** is *DownloadFailed*	n/a
Expected result(s) / behaviour	n/a	n/a

3.15.3. Firmware Update - Installation Failed

Table 158. Test Case Id: TC_044_3_CSMS

Test case name	**Firmware Update - Installation Failed**	
Test case Id	TC_044_3_CSMS	
Description	The firmware of a Charge Point is being updated, but the installation fails.	
Purpose	Check whether Central System can handle messages for an update of the firmware of a Charge Point in case the installation fails.	
Prerequisite(s)	The Central System supports the Firmware Management feature profile	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **UpdateFirmware.conf**	**1.** The Central System sends a **UpdateFirmware.req**
	[The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **FirmwareStatusNotification.req**	**4.** The Central responds with a **FirmwareStatusNotification.conf**
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **FirmwareStatusNotification.req**	**6.** The Central responds with a **FirmwareStatusNotification.conf**
	[The Charge Point reports the status of all connectors] **7.** The Charge Point sends a **StatusNotification.req**	**8.** The Central responds with a **StatusNotification.conf**
	[The Charge Point starts installing the firmware] **9.** The Charge Point sends a **FirmwareStatusNotification.req**	**10.** The Central responds with a **FirmwareStatusNotification.conf**
	11. The Charge point reboots and sends a **BootNotification.req**	**12.** The Central System responds with a **BootNotification.conf**
	[The Charge Point reports the status of all connectors] **13.** The Charge Point sends a **StatusNotification.req**	**14.** The Central responds with a **StatusNotification.conf**
	15. The Charge Point sends a **FirmwareStatusNotification.req**	**16.** The Central responds with a **FirmwareStatusNotification.conf**

Test case name	**Firmware Update - Installation Failed**	
Tool validation(s)	* Step 3: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **FirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (Message: **StatusNotification.req**) The **status** is *Unavailable* * Step 9: (Message: **FirmwareStatusNotification.req**) The **status** is *Installing* * Step 13: (Message: **StatusNotification.req**) The **status** is *Available* * Step 15: (Message: **FirmwareStatusNotification.req**) The **status** is *InstallationFailed*	n/a
Expected result(s) / behaviour	n/a	n/a

3.16. Diagnostics

3.16.1. Get Diagnostics

Table 159. Test Case Id: TC_045_1_CSMS

Test case name	**Get Diagnostics**	
Test case Id	TC_045_1_CSMS	
Description	The Charge Point uploads a diagnostics log to a specified location based on a request of the Central System.	
Purpose	The purpose of this test case it to check whether Central System can trigger the Charge Point to upload its diagnostics.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetDiagnostics.conf** to the Central System.	**1.** The Central System sends a **GetDiagnostics.req** to the Charge Point.
	[The Charge Point starts uploading the diagnostics log.] **3.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**4.** The Central responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.
	[The Charge Point has finished uploading the diagnostics log.] **5.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**6.** The Central responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.

Test case name	**Get Diagnostics**	
Tool validation(s)	* Step 3: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Uploading* * Step 5: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Uploaded*	n/a
Expected result(s) / behaviour	The Charge Point has uploaded the diagnostics log to the **location** that was sent in step 1.	n/a

3.16.2. Get Diagnostics - Upload Failed

Table 160. Test Case Id: TC_045_2_CSMS

Test case name	**Get Diagnostics - Upload Failed**	
Test case Id	TC_045_2_CSMS	
Description	When getting the diagnostics of a Charge Point, the upload of the log fails.	
Purpose	Check whether Central System can handle messages for the situation that the upload fails when getting the diagnostics.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetDiagnostics.conf** to the Central System.	**1.** The Central System sends a **GetDiagnostics.req** to the Charge Point.
	[The Charge Point starts uploading the diagnostics log.] **3.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**4.** The Central responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.
	[The Charge Point has failed uploading the diagnostics log.] **5.** The Charge Point sends a **DiagnosticsStatusNotification.req** to the Central System.	**6.** The Central responds with a **DiagnosticsStatusNotification.conf** to the Charge Point.
Tool validation(s)	* Step 3: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Uploading* * Step 5: (Message: **DiagnosticsStatusNotification.req**) The **status** is *UploadFailed*	n/a
Expected result(s) / behaviour	The Charge Point continues normal operation.	n/a

3.17. Reservation

3.17.1. Reservation of a Connector

Reservation of a Connector - Transaction

Table 161. Test Case Id: TC_046_CSMS

Test case name	**Reservation of a Connector - Transaction**	
Test case Id	TC_046_CSMS	
Description	A Connector is reserved and a charging transaction takes place.	
Purpose	Check whether Central System can trigger a Charge Point to Reserve a Connector.	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
	3 The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	5. Execute **Reusable State** *Charging*	
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) - The **status** is *Reserved* * Step 5: (Reusable State: **Charging**) - The **reservationId** is the reservationId from step 1	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** should be *<Configured ConnectorId>* - The **idTag** should be *<Configured Valid IdTag>*
Expected result(s) / behaviour	n/a	n/a

Reservation of a Connector - Expire

Table 162. Test Case Id: TC_047_CSMS

Test case name	**Reservation of a Connector - Expire**
Test case Id	TC_047_CSMS
Description	A Connector is reserved, a charging transaction could take place, but the reservation is not used (in time)
Purpose	Check whether Central System can handle messages when the reservation is not used (in time).
Prerequisite(s)	The Central System supports the Reservation feature profile.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Reservation of a Connector - Expire**	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
	3 The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	[EV driver does not arrive at the reserved Connector before the expiry date] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) - The **status** is *Reserved* * Step 5: (Message: **StatusNotification.req**) - The **status** is *Available*	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** should be *<Configured ConnectorId>* - The **idTag** should be *<Configured Valid IdTag>* - The **expiryDate** should be the current time plus *<Configured Expiry Date Offset>*
Expected result(s) / behaviour	n/a	n/a

Reservation of a Connector - Faulted

Table 163. Test Case Id: TC_048_1_CSMS

Test case name	**Reservation of a Connector - Faulted**	
Test case Id	TC_048_1_CSMS	
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Faulted* is returned by the Charge Point.	
Purpose	Check whether the Central System is able to handle messages in case that a reservation cannot be made.	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Faulted*	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** should be *<Configured ConnectorId>* - The **idTag** should be *<Configured Valid IdTag>*
Expected result(s) / behaviour	n/a	The Central System accepts the Reservation message with the not *Accepted* status.

Reservation of a Connector - Occupied

Table 164. Test Case Id: TC_048_2_CSMS

Test case name	**Reservation of a Connector - Occupied**
Test case Id	TC_048_2_CSMS
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Occupied* is returned by the Charge Point.

Test case name	**Reservation of a Connector - Occupied**	
Purpose	Check whether the Central System can handle messages in case that a reservation cannot be made.	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	[EV driver plugs in cable] **1.** The Charge Point sends a **StatusNotification.req**	**2.** The Central System responds with a **StatusNotification.conf**
	4. The Charge Point responds with a **ReserveNow.conf**	**3.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 1: (Message: **StatusNotification.req**) - The **status** is *Preparing* - The **connectorId** is *<Configured ConnectorId>* * Step 4: (Message: **ReserveNow.conf**) - The **status** is *Occupied*	* Step 3: (Message: **ReserveNow.req**) - The **connectorId** should be the **connectorId** from step 1. - The **idTag** should be *<Configured Valid IdTag>*
Expected result(s) / behaviour	n/a	The Central System accepts the Reservation message with the not *Accepted* status.

Reservation of a Connector - Unavailable

Table 165. Test Case Id: TC_048_3_CSMS

Test case name	**Reservation of a Connector - Unavailable**	
Test case Id	TC_048_3_CSMS	
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Unavailable* is returned by the Charge Point.	
Purpose	Check whether the Central System can handle messages in case that a reservation cannot be made.	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ChangeAvailability.conf**	**1.** The Central System sends a **ChangeAvailability.req**
	3. The Charge Point sends a **StatusNotification.req**	**4.** The Central System responds with a **StatusNotification.conf**
	6. The Charge Point responds with a **ReserveNow.conf**	**5.** The Central System sends a **ReserveNow.req**

Test case name	Reservation of a Connector - Unavailable	
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) - The **Status** is *Unavailable* - The **connectorId** equals the **connectorId** from step 1. * Step 6: (Message: **ReserveNow.conf**) - The **status** is *Unavailable*	* Step 1: (Message: **ChangeAvailability.req**) - The **connectorId** should be *<Configured ConnectorId>* - The **type** is *Inoperative* * Step 5: (Message: **ReserveNow.req**) - The **connectorId** should be the **connectorId** from step 1. - The **idTag** should be *<Configured Valid IdTag>*
Expected result(s) / behaviour	n/a	The Central System accepts the Reservation message with the not *Accepted* status.

Reservation of a Connector - Rejected

Table 166. Test Case Id: TC_048_4_CSMS

Test case name	Reservation of a Connector - Rejected	
Test case Id	TC_048_4_CSMS	
Description	The Central System attempts to reserve a Connector, but the reservation is not made, instead the status *Rejected* is returned by the Charge Point.	
Purpose	Check whether the Central System can handle messages in case that a reservation cannot be made.	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ReserveNow.conf**	**1.** The Central System sends a **ReserveNow.req**
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) - The **status** is *Rejected*	* Step 1: (Message: **ReserveNow.req**) - The **connectorId** should be *<Configured ConnectorId>* - The **idTag** should be *<Configured Valid IdTag>*
Expected result(s) / behaviour	n/a	The Central System accepts the Reservation message with the not *Accepted* status.

3.17.2. Reservation of a Charge Point

Reservation of a Charge Point - Transaction

Table 167. Test Case Id: TC_049_CSMS

Test case name	Reservation of a Charge Point - Transaction
Test case Id	TC_049_CSMS
Description	A Charge Point / unspecified Connector is reserved and a charging transaction takes place.
Purpose	Check whether Central System trigger the Charge Point to reserve an unspecified Connector.
Prerequisite(s)	The Central System supports the Reservation feature profile.

Test case name	**Reservation of a Charge Point - Transaction**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point sends a **ReserveNow.conf** message to the Central System	**1.** The Central System sends a **ReserveNow.req** with a *reservationId*, *connectorId* and *idTag* to the Charge Point
	3 The Charge Point sends a **StatusNotification.req** to the Central System	**4.** The Central System sends a **StatusNotification.conf** to the Charge Point
Tool validation(s)	* Step 3: (Message: **StatusNotification.req**) The **status** is *Reserved*	* Step 1: (Message: **ReserveNow.req**) The **connectorId** is *0*
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, only the **idTag** from the reservation can charge, on any available connector of the Charge Point.	The Central System accepts the reservation for the right **idTag** and **reservationId**.

3.17.3. Cancel Reservation

Cancel Reservation

Table 168. Test Case Id: TC_051_CSMS

Test case name	**Cancel Reservation**	
Test case Id	TC_051_CSMS	
Description	The Central System cancels an existing, not expired reservation.	
Purpose	Check whether the Central System trigger to Charge Point to cancel a reservation.	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point sends a **ReserveNow.conf** message to the Central System	**1.** The Central System sends a **ReserveNow.req** with a *reservationId*, *connectorId*, *idTag* and *expiryDate* to the Charge Point
	3. The Charge Point sends a **StatusNotification.req** to the Central System	**4.** The Central System sends a **StatusNotification.conf** to the Charge Point
	6. The Charge Point sends a **CancelReservation.conf** message to the Central System	**5.** The Central System sends a **CancelReservation.req** with a *reservationId* to the Charge Point
	7. The Charge Point sends a **StatusNotification.req** to the Central System	**8.** The Central System sends a **StatusNotification.conf** to the Charge Point

Test case name	**Cancel Reservation**	
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) The **status** is *Reserved* * Step 6: (Message: **CancelReservation.conf**) The **status** is *Accepted* * Step 7: (Message: **StatusNotification.req**) The **status** is *Available*	* Step 1: (Message: **ReserveNow.req**) The **connectorId** does not equal *0* * Step 5: (Message: **CancelReservation.req**) The **reservationId** matches the **reservationId** from step 1.
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, cancelling only the reservation with the right reservationId.	The Central System processes the response from the Charge Point to the cancel reservation message.

Cancel Reservation - Rejected

Table 169. Test Case Id: TC_052_CSMS

Test case name	**Cancel Reservation - Rejected**	
Test case Id	TC_052_CSMS	
Description	The Central System tries to cancel reservation, but this request is rejected by the Charge Point.	
Purpose	Check whether the Central System can handle messages in case cancelling a reservation is rejected by the Charge Point.	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	Manual Action: *Reserve a connector on the Charge Point.*	
	2. The Charge Point sends a **ReserveNow.conf**.	**1.** The Central System sends a **ReserveNow.req**.
	3. The Charge Point sends a **StatusNotification.req**.	**4.** The Central System sends a **StatusNotification.conf**.
	Manual Action: *Cancel the reservation on the Charge Point.*	
	6. The Charge Point sends a **CancelReservation.conf**.	**5.** The Central System sends a **CancelReservation.req** with an unknown *reservationId*.
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) The **status** is *Reserved* * Step 6: (Message: **CancelReservation.conf**) The **status** is *Rejected*	* Step 5: (Message: **CancelReservation.req**) The **reservationId** does NOT match the **reservationId** from step 1.
Expected result(s) / behaviour	The Charge Point rejects the unknown *reservationId* and does not cancel any reservation.	The Central System processes the rejection from the Charge Point to the cancel reservation message.

3.17.4. Use a reserved Connector with parentIdTag

Table 170. Test Case Id: TC_053_CSMS

Test case name	**Use a reserved Connector with parentIdTag**	
Test case Id	TC_053_CSMS	
Description	The Charge Point has been reserved and is used with a *parentIdTag*	
Purpose	Check whether the Central System can handle messages for a reservation that is used by a *parentIdTag*	
Prerequisite(s)	The Central System supports the Reservation feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	Manual Action: *Reserve a connector on the Charge Point.*	
	2. The Charge Point sends a **ReserveNow.conf**.	**1.** The Central System sends a **ReserveNow.req** with a *reservationId*, an *idTag* and a *parentIdTag*.
	3. The Charge Point sends a **StatusNotification.req**.	**4.** The Central System sends a **StatusNotification.conf**.
	5. Execute reusable state *Charging*	
	[EV driver authorizes / swipes card with the parentIdTag from step 1, but a new IdTag] **6.** The Charge Point sends an **Authorize.req**.	**7.** The Central System sends an **Authorize.conf**.
	8. The Charge Point sends a **StopTransaction.req**.	**9.** The Central System sends a **StopTransaction.conf**.
	10. The Charge Point sends a **StatusNotification.req**.	**11.** The Central System sends a **StatusNotification.conf**.
Tool validation(s)	* Step 2: (Message: **ReserveNow.conf**) The **status** is *Accepted* * Step 3: (Message: **StatusNotification.req**) The **status** is *Reserved* * Step 6: (Message: **Authorize.req**) The **idTag** is different from step 1 and 7. * Step 10: (Message: **StatusNotification.req**) The **status** is *Finishing*	* Step 1: (Message: **ReserveNow.req**) The **connectorId** does not equal *0* * Step 7: (Message: **Authorize.conf**) The **idTagInfo.status** is *Accepted* The **idTagInfo.parentIdTag** matches the **parentIdTag** from step 1
Expected result(s) / behaviour	The Charge Point handles the reservation correctly, the **parentIdTag** from the reservation can charge on the reserved Connector.	The Central System accepts the reservation for the right **parentIdTag** and **reservationId**.

3.18. RemoteTrigger

3.18.1. Trigger Message

Table 171. Test Case Id: TC_054_CSMS

Test case name	**Trigger Message**
Test case Id	TC_054_CSMS
Description	The Central System triggers a message from the Charge Point
Purpose	Check whether the Central System is able to trigger a message from the Charge Point.

Test case name	**Trigger Message**	
Prerequisite(s)	The Central System supports the Remote Trigger feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **TriggerMessage.conf**	**1.** The Central System sends a **TriggerMessage.req**
	3. The Charge Point sends a **MeterValues.req**	**4.** The Central System responds with a **MeterValues.conf**
	6. The Charge Point responds with a **TriggerMessage.conf**	**5.** The Central System sends a **TriggerMessage.req**
	7. The Charge Point sends a **Heartbeat.req**	**8.** The Central System responds with a **Heartbeat.conf**
	10. The Charge Point responds with a **TriggerMessage.conf**	**9.** The Central System sends a **TriggerMessage.req**
	11. The Charge Point sends a **StatusNotification.req**	**12.** The Central System responds with a **StatusNotification.conf**
	14. The Charge Point responds with a **TriggerMessage.conf**	**13.** The Central System sends a **TriggerMessage.req**
	15. The Charge Point sends a **DiagnosticsStatusNotification.req**	**16.** The Central System responds with a **DiagnosticsStatusNotification.conf**
	18. The Charge Point responds with a **TriggerMessage.conf**	**17.** The Central System sends a **TriggerMessage.req**
	[The following message will be sent if implemented.] **19.** The Charge Point sends a **FirmwareStatusNotification.req**	**20.** The Central System responds with a **FirmwareStatusNotification.conf**
Tool validation(s)	* Step 2/6/10/14: (Message: **TriggerMessage.conf**) The **status** is *Accepted* * Step 15: (Message: **DiagnosticsStatusNotification.req**) The **status** is *Idle* * Step 18: (Message: **TriggerMessage.conf**) The **status** is *Accepted* OR *NotImplemented* * Step 19: (Message: **FirmwareStatusNotification.req**) The **status** is *Idle*	* Step 1: (Message: **TriggerMessage.req**) **requestedMessage** should be *MeterValues* **connectorId** should be *<Configured ConnectorId>* * Step 5: (Message: **TriggerMessage.req**) **requestedMessage** should be *Heartbeat* * Step 9: (Message: **TriggerMessage.req**) **requestedMessage** should be *StatusNotification* **connectorId** should be *<Configured ConnectorId>* * Step 13: (Message: **TriggerMessage.req**) **requestedMessage** should be *DiagnosticsStatusNotification* * Step 17: (Message: **TriggerMessage.req**) **requestedMessage** should be *FirmwareStatusNotification*
Expected result(s) / behaviour	n/a	The Central System can request a message from a Charge Point and receive the requested message.

3.18.2. Trigger Message - Rejected

Table 172. Test Case Id: TC_055_CSMS

Test case name	**Trigger Message - Rejected**	
Test case Id	TC_055_CSMS	
Description	The Central System triggers a message from the Charge Point, but the Charge Point rejects the message.	
Purpose	To check whether the Central System is able to handle a reject on a triggered message.	
Prerequisite(s)	- The Central System supports the Remote Trigger feature profile. - The Central System supports sending a **TriggerMessage.req** for a non-configured connector.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **TriggerMessage.conf**	**1.** The Central System sends a **TriggerMessage.req**
Tool validation(s)	* Step 2: (Message: **TriggerMessage.conf**) The **status** is *Rejected*	* Step 1: (Message: **TriggerMessage.req**) The **requestMessage** should be *MeterValues* The **connectorId** should be *<Configured NumberOfConnectors + 1>*
Expected result(s) / behaviour	n/a	The Central System processes the response from the Charge Point.

3.19. SmartCharging

3.19.1. Central Smart Charging

Central Smart Charging - TxDefaultProfile

Table 173. Test Case Id: TC_056_CSMS

Test case name	**Central Smart Charging - TxDefaultProfile**	
Test case Id	TC_056_CSMS	
Description	The Central System sets a default schedule for new transactions.	
Purpose	To check whether the Central System can set a default schedule for new transactions.	
Prerequisite(s)	The Central System supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **SetChargingProfile.conf**	**1.** The Central System sends a **SetChargingProfile.req**

Test case name	**Central Smart Charging - TxDefaultProfile**	
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) **status** is *Accepted*	* Step 1: (Message: **SetChargingProfile.req**) **connectorId** *<Configured connectorId>* AND **csChargingProfiles.stackLevel** *<Configured stackLevel>* AND **csChargingProfiles.chargingProfilePurpose** *TxDefaultProfile* AND **csChargingProfiles.chargingProfileKind** *Absolute* AND **csChargingProfiles.validFrom** *<Not omitted>* AND **csChargingProfiles.validTo** *<Not omitted>* AND **csChargingProfiles.chargingSchedule.startSchedule** *<Not omitted>* AND **csChargingProfiles.chargingSchedule.chargingRateUnit** *<Configured chargingRateUnit>* AND **csChargingProfiles.chargingSchedule.duration** *<Configured duration>* AND **csChargingProfiles.chargingSchedule.chargingSchedulePeriod.startPeriod** *<Configured startPeriod>* AND **csChargingProfiles.chargingSchedule.chargingSchedulePeriod.limit** *6.0* or *6000.0* AND **csChargingProfiles.chargingSchedule.chargingSchedulePeriod.numberPhases** *<Configured numberPhases>* where *<Configured numberPhases>* not *3* OR **csChargingProfiles.chargingSchedule.chargingSchedulePeriod.numberPhases** *<Configured numberPhases>* or *<omit>* where *<Configured numberPhases> 3*
Expected result(s) / behaviour	n/a	n/a

Central Smart Charging - TxProfile

Table 174. Test Case Id: TC_057_CSMS

Test case name	**Central Smart Charging - TxProfile**	
Test case Id	TC_057_CSMS	
Description	The Central System sets a schedule for a running transaction.	
Purpose	To check whether the Central System is able to set a schedule for a running transaction on a Charge Point.	
Prerequisite(s)	The Central System supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Charging*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **SetChargingProfile.conf**	**1.** The Central System sends a **SetChargingProfile.req**
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) **status** is *Accepted*	* Step 1: (Message: **SetChargingProfile.req**) **connectorId** *<Configured connectorId>* AND **csChargingProfiles.ChargingProfilePurpose** *TxProfile* AND **csChargingProfiles.transactionId** *<Generated transactionId>*

Test case name	**Central Smart Charging - TxProfile**	
Expected result(s) / behaviour	n/a	n/a

3.19.2. Get Composite Schedule

Table 175. Test Case Id: TC_066_CSMS

Test case name	**Get Composite Schedule**	
Test case Id	TC_066_CSMS	
Description	The Central System requests a composite schedule.	
Purpose	To check whether the Central System is able to request a composite schedule.	
Prerequisite(s)	The Central System supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetCompositeSchedule.conf**	**1.** The Central System sends a **GetCompositeSchedule.req**
Tool validation(s)	* Step 2: (Message: **GetCompositeSchedule.conf**) - **chargingSchedule** contains a hard-coded composite schedule.	* Step 1: (Message: **GetCompositeSchedule.req**) - **connectorId** should be *<Configured ConnectorId>* - **duration** should be *<Configured Charging Schedule Duration>* - **chargingRateUnit** should be *<Configured Charging Rate Unit>*
Expected result(s) / behaviour	n/a	The Central System has retrieved the composite *ChargingProfile*.

3.19.3. Clear Charging Profile

Table 176. Test Case Id: TC_067_CSMS

Test case name	**Clear Charging Profile**
Test case Id	TC_067_CSMS
Description	The Central Systems sets a Charging Profile and clears it.
Purpose	To check whether the Central System can clear a charging profile.
Prerequisite(s)	The Central System supports the Smart Charging feature profile.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Clear Charging Profile**	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	Manual Action: Set three different charging profiles. Steps 1-2 are therefor repeated three times.	
	2. The Charge Point responds with a **SetChargingProfile.conf**	**1.** The Central System sends a **SetChargingProfile.req**
	Manual Action: Clear a charging profile based on ID.	
	4. The Charge Point responds with a **ClearChargingProfile.conf**	**3.** The Central System sends a **ClearChargingProfile.req**
	Manual Action: Clear a charging profile based on criteria.	
	6. The Charge Point responds with a **ClearChargingProfile.conf**	**5.** The Central System sends a **ClearChargingProfile.req**
	Manual Action: Clear all remaining charging profiles.	
	8. The Charge Point responds with a **ClearChargingProfile.conf**	**7.** The Central System sends a **ClearChargingProfile.req**

Test case name	**Clear Charging Profile**	
Tool validation(s)	* Step 2: (Message: **SetChargingProfile.conf**) - The **status** is *Accepted* * Step 4/6/8: (Message: **ClearChargingProfile.conf**) - The **status** is *Accepted*	* Step 1: (Message: **SetChargingProfile.req**) Charging profile 1: - The **connectorId** should be *<Configured ConnectorId>* - The **chargingProfilePurpose** should be *TxDefaultProfile* - The **stackLevel** should be <Configured Stack Level> Charging profile 2: - The **connectorId** should be *<Configured ConnectorId>* - The **chargingProfilePurpose** should be *TxDefaultProfile* - The **stackLevel** should be <Configured Stack Level + 1> Charging profile 3: - The **connectorId** should be *<Configured ConnectorId>* - The **chargingProfilePurpose** should be *TxDefaultProfile* - The **stackLevel** should be <Configured Stack Level + 2> * Step 3: (Message: **ClearChargingProfile.req**) - The **id** should be <Generated Id from charging profile 1> - The **connectorId**, **chargingProfilePurpose** and **stackLevel** fields should be omitted. * Step 5: (Message: **ClearChargingProfile.req**) - The **id** should be omitted - The **connectorId** should be *<Configured ConnectorId>* - The **chargingProfilePurpose** should be *TxDefaultProfile* - The **stackLevel** should be *<Configured Stack Level + 1>* * Step 7: (Message: **ClearChargingProfile.req**) - All fields should be omitted.
Expected result(s) / behaviour	n/a	The Central System was able to clear the *ChargingProfile* of the Charge Point.

3.19.4. Remote Start Transaction with Charging Profile

Remote Start Transaction with Charging Profile

Table 177. Test Case Id: TC_059_CSMS

Test case name	Remote Start Transaction with Charging Profile	
Test case Id	TC_059_CSMS	
Description	The Central System starts a transaction on a Charge Point with a *ChargingProfile*	
Purpose	To check whether the Central System can trigger a Charge Point to start a transaction with a Charging Profile.	
Prerequisite(s)	The Central System supports the Smart Charging feature profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **RemoteStartTransaction.conf**	**1.** The Central Systems sends a **RemoteStartTransaction.req**
	3. The Charge Point sends an **Authorize.req**	**4.** The Central System responds with an **Authorize.conf**
	[The charging cable is plugged in] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
	7. The Charge Point sends a **StartTransaction.req**	**8.** The Central System responds with a **StartTransaction.conf**
	9. The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **RemoteStartTransaction.conf**) - The **status** is *Accepted* * Step 3: (Message: **Authorize.req**) - The **idTag** is the idTag from step 1. * Step 5: (Message: **StatusNotification.req**) - The **status** is *Preparing* - The **connectorId** is the connectorId from step 1. * Step 7: (Message: **StartTransaction.req**) - The **idTag** is the idTag from step 1. - The **connectorId** is the connectorId from step 1. * Step 9: (Message: **StatusNotification.req**) - The **status** is *Charging* - The **connectorId** is the connectorId from step 1.	* Step 1: (Message: **RemoteStartTransaction.req**) - The **idTag** is *<Configured valid IdTag>* - The **connectorId** is *<Configured ConnectorId>* - The **chargingProfile.chargingProfilePurpose** is *TxProfile* - The **chargingProfile.transactionId** is omitted - The **chargingProfile.chargingProfileKind** is *Relative* - The **chargingProfile.chargingSchedule.chargingSchedulePeriod.startPeriod** is *0* * Step 4: (Message: **Authorize.conf**) - The **idTagInfo.status** is *Accepted* * Step 8: (Message: **StartTransaction.conf**) - The **status** is *Accepted*
Expected result(s) / behaviour	n/a	The Central System has started a transaction on the Charge Point and accepts the transaction that is started on the Charge Point.

3.20. DataTransfer

3.20.1. Data Transfer to a Central System

Table 178. Test Case Id: TC_064_CSMS

Test case name	**Data Transfer to a Central System**	
Test case Id	TC_064_CSMS	
Description	The Charge Point sends a vendor specific message to the Central System.	
Purpose	To check whether the Central System can reject vendor specific messages.	
Prerequisite(s)	The Central System does not support DataTransfer for a specific *vendorId*.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends a **DataTransfer.req** message with a specific *vendorId* to the Charge Point.	**2.** The Central System responds with a **DataTransfer.conf** message.
Tool validation(s)	n/a	* Step 2: (Message: **DataTransfer.conf**) The **status** is *Rejected* OR *UnknownMessageId* OR *UnknownVendorId* **Note:** The **status** *Accepted* is allowed, but the vendor should be warned about this behaviour.
Expected result(s) / behaviour	n/a	The Central System does not accept the **DataTransfer.req**.

3.21. Security

3.21.1. Secure connection setup

Update Charge Point Password for HTTP Basic Authentication

Table 179. Test Case Id: TC_073_CSMS

Test case name	**Update Charge Point Password for HTTP Basic Authentication**	
Test case Id	TC_073_CSMS	
Description	The Central System can configure a new password for HTTP Basic Authentication, the Central System can send a new value for the BasicAuthPassword Configuration key.	
Purpose	To check if the Central System is able to change the Basic Authentication password.	
Prerequisite(s)	The Central System supports Security profile 1 and/or 2.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	Manual Action: *Change the basic authentication password.*	
	2. The Charge Point responds with a **ChangeConfiguration.conf**	**1.** The Central System sends a **ChangeConfiguration.req**
	3. The Charge Point disconnects its current connection and reconnects to the Central System with the new password.	

Test case name	**Update Charge Point Password for HTTP Basic Authentication**	
Tool validation(s)	* Step 2: (Message: **ChangeConfiguration.conf**) **status** is *Accepted* * Step 3: *The Charge Point reconnects to the Central System with the new password.*	* Step 1: (Message: **ChangeConfiguration.req**) **key** is *AuthorizationKey*
Expected result(s) / behaviour	n/a	n/a

Update Charge Point Certificate by request of Central System

Table 180. Test Case Id: TC_074_CSMS

Test case name	**Update Charge Point Certificate by request of Central System**	
Test case Id	TC_074_CSMS	
Description	When SUT Charge Point, the tool shall take on the role of both Central System and Certificate Authority Server. Which means it will sign the certificate with its own certificate.	
Purpose	To check if the Central System is able to request the Charge Point to renew its ChargePointCertificate.	
Prerequisite(s)	The Central System supports security profile 3.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ExtendedTriggerMessage.conf**	**1.** The Central System sends a **ExtendedTriggerMessage.req**
	[The Charge Point generates a new public/private key pair and generates a Certificate Signing Request.] **3.** The Charge Point sends a **SignCertificate.req**.	**4.** The Central System responds with a **SignCertificate.conf**.
	[The Charge Point verifies the validity of the signed certificate.] **6.** The Charge Point responds with a **CertificateSigned.conf**.	[Certificate Authority Server signs the certificate.] **5.** The Central System sends a **CertificateSigned.req**.
	7. The Charge Point disconnects its current connection and reconnects to the Central System with the new certificate.	
Tool validation(s)	* Step 2: (Message: **ExtendedTriggerMessage.conf**) The **status** is *Accepted* * Step 6: (Message: **CertificateSigned.conf**) The **status** is *Accepted* * Step 7: *The Charge Point reconnects to the Central System with the new certificate.*	* Step 1: (Message: **ExtendedTriggerMessage.req**) The **requestedMessage** is *SignChargePointCertificate* The **connectorId** is *<Omitted>* * Step 4: (Message: **SignCertificate.conf**) The **status** is *Accepted*
Expected result(s) / behaviour	n/a	n/a

Install a certificate on the Charge Point - ManufacturerRootCertificate

Table 181. Test Case Id: TC_075_1_CSMS

Test case name	**Install a certificate on the Charge Point - ManufacturerRootCertificate**
Test case Id	TC_075_1_CSMS

Test case name	**Install a certificate on the Charge Point - ManufacturerRootCertificate**	
Description	The Central System requests the Charge Point to install a new Manufacturer root certificate.	
Purpose	To check if the Central System is able to install a certificate on the Charge Point.	
Prerequisite(s)	The Central System supports Security profile 2 and/or 3.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **InstallCertificate.conf**	**1.** The Central System sends a **InstallCertificate.req**
	4. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**3.** The Central System sends a **GetInstalledCertificateIds.req**
Tool validation(s)	* Step 2: (Message: **InstallCertificate.conf**) **status** is *Accepted* * Step 4: (Message: **GetInstalledCertificateIds.conf**) The **status** is *Accepted* **certificateHashData** is *<Includes the certificate information of the installed certificate from step 1.>* **Note:** This test case must be executed with a Root CA certificate in order to get the correct response message from the OCTT.	* Step 1: (Message: **InstallCertificate.req**) **certificateType** is *ManufacturerRootCertificate* **certificate** is *<Configured root certificate>* * Step 3: (Message: **GetInstalledCertificateIds.req**) The **certificateType** is *ManufacturerRootCertificate*
Expected result(s) / behaviour	n/a	n/a

Install a certificate on the Charge Point - CentralSystemRootCertificate

Table 182. Test Case Id: TC_075_2_CSMS

Test case name	**Install a certificate on the Charge Point - CentralSystemRootCertificate**	
Test case Id	TC_075_2_CSMS	
Description	The Central System requests the Charge Point to install a new Central System root certificate.	
Purpose	To check if the Central System is able to install a certificate on the Charge Point.	
Prerequisite(s)	The Central System supports Security profile 2 and/or 3.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **InstallCertificate.conf**	**1.** The Central System sends a **InstallCertificate.req**
	4. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**3.** The Central System sends a **GetInstalledCertificateIds.req**

Test case name	**Install a certificate on the Charge Point - CentralSystemRootCertificate**	
Tool validation(s)	* Step 2: (Message: **InstallCertificate.conf**) **status** is *Accepted* * Step 4: (Message: **GetInstalledCertificateIds.conf**) The **status** is *Accepted* **certificateHashData** is *<Includes the certificate information of the installed certificate from step 1.>* **Note:** This test case must be executed with a Root CA certificate in order to get the correct response message from the OCTT.	* Step 1: (Message: **InstallCertificate.req**) **certificateType** is *CentralSystemRootCertificate* **certificate** is *<Configured root certificate>* * Step 3: (Message: **GetInstalledCertificateIds.req**) The **certificateType** is *CentralSystemRootCertificate*
Expected result(s) / behaviour	n/a	n/a

Delete a specific certificate from the Charge Point

Table 183. Test Case Id: TC_076_CSMS

Test case name	**Delete a specific certificate from the Charge Point**	
Test case Id	TC_076_CSMS	
Description	To facilitate the management of the Charge Point’s installed certificates, a method of deleting an installed certificate is provided. The Central System requests the Charge Point to delete a specific certificate.	
Purpose	To check if the Central System is able to delete an installed certificate from the Charge Point.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): - Request Central System to install a CentralSystemRootCertificate (Root 2)	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	The OCTT requests the Central System to delete the just installed CentralSystemRootCertificate	
	2. The Charge Point responds with a **GetInstalledCertificateIds.conf**	**1.** The Central System sends a **GetInstalledCertificateIds.req**
	4. The Charge Point responds with a **DeleteCertificate.conf**	**3.** The Central System sends a **DeleteCertificate.req**
Tool validation(s)	* Step 2: (Message: **GetInstalledCertificateIds.conf**) **status** is *Accepted* **certificateHashData.hashAlgorithm** is *<Configured HashAlgorithm>* * Step 4: (Message: **DeleteCertificate.conf**) **status** is *Accepted*	* Step 1: (Message: **DeleteCertificate.req**) **hashAlgorithm** is *<Configured HashAlgorithm>* (It needs to be equal to the hashAlgorithm returned at step 2) **certificateHashData** is *<Includes the certificate information of the installed CentralSystemRootCertificate.>* The individual fields of the **certificateHashData** are verified by the OCTT (the OCTT compares these with its own **certificateHashData** calculation).
Expected result(s) / behaviour	n/a	n/a

3.21.2. Security event/logging

Invalid ChargePointCertificate Security Event

Table 184. Test Case Id: TC_077_CSMS

Test case name	**Invalid ChargePointCertificate Security Event**	
Test case Id	TC_077_CSMS	
Description	The Charge Point notifies the Central System of an invalid certificate.	
Purpose	To check if the Central System can handle when a Charge Point registers a security event and notifies the Central System about it.	
Prerequisite(s)	The Central System supports security profile 3.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **ExtendedTriggerMessage.conf**	**1.** The Central System sends a **ExtendedTriggerMessage.req**
	[The Charge Point generates a new public/private key pair and generates a Certificate Signing Request.] **3.** The Charge Point sends a **SignCertificate.req**.	**4.** The Central System responds with a **SignCertificate.conf**.
	[The Charge Point verifies the validity of the signed certificate.] **6.** The Charge Point responds with a **CertificateSigned.conf**.	**5.** The Central System sends a **CertificateSigned.req**.
	7. The Charge Point sends a **SecurityEventNotification.req**	**8.** The Central System responds with a **SecurityEventNotification.conf**
Tool validation(s)	* Step 2: (Message: **ExtendedTriggerMessage.conf**) The **status** is *Accepted* * Step 6: (Message: **CertificateSigned.conf**) The **status** is *Rejected* * Step 7: (Message: **SecurityEventNotification.req**) The **type** is *InvalidChargePointCertificate*	* Step 1: (Message: **ExtendedTriggerMessage.req**) The **requestedMessage** is *SignChargePointCertificate* The **connectorId** is *<Omitted>* * Step 4: (Message: **SignCertificate.conf**) The **status** is *Accepted* * Step 5: (Message: **CertificateSigned.req**) The **certificate** is *<Signed ChargePointCertificate>*
Expected result(s) / behaviour	n/a	n/a

Invalid CentralSystemCertificate Security Event

Table 185. Test Case Id: TC_078_CSMS

Test case name	**Invalid CentralSystemCertificate Security Event**
Test case Id	TC_078_CSMS
Description	The Charge Point notifies the Central System of an invalid certificate.
Purpose	To check if the Central System can handle it when a Charge Point registers a security event and notifies the Central System about it.
Prerequisite(s)	The Central System supports Security profile 2 and/or 3.

Test case name	**Invalid CentralSystemCertificate Security Event**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with an **InstallCertificate.conf**	**1.** The Central System sends an **InstallCertificate.req**
	3. The Charge Point sends a **SecurityEventNotification.req**	**4.** The Central System responds with a **SecurityEventNotification.conf**
Tool validation(s)	* Step 2: (Message: **InstallCertificate.conf**) **status** is *Rejected* * Step 3: (Message: **SecurityEventNotification.req**) The **type** is *InvalidCentralSystemCertificate*	* Step 1: (Message: **InstallCertificate.req**) **certificateType** is *CentralSystemRootCertificate* **certificate** is *<Configured certificate>* **Note**: For this testcase he OCTT will reject any certificate.
Expected result(s) / behaviour	n/a	n/a

Get Security Log

Table 186. Test Case Id: TC_079_CSMS

Test case name	**Get Security Log**	
Test case Id	TC_079_CSMS	
Description	The Charge Point uploads a security log to a specified location based on a request of the Central System.	
Purpose	To check whether Central System can trigger a Charge Point to upload its security log.	
Prerequisite(s)	The Central System supports a security profile.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point responds with a **GetLog.conf**.	**1.** The Central System sends a **GetLog.req**.
	[The Charge Point starts uploading the security log.] **3.** The Charge Point sends a **LogStatusNotification.req**.	**4.** The Central System responds with a **LogStatusNotification.conf**.
	[The Charge Point has finished uploading the security log.] **5.** The Charge Point sends a **LogStatusNotification.req**.	**6.** The Central System responds with a **LogStatusNotification.conf**.
Tool validation(s)	* Step 2: (Message: **GetLog.conf**) The **status** is *Accepted* * Step 3: (Message: **LogStatusNotification.req**) The **status** is *Uploading* * Step 5: (Message: **LogStatusNotification.req**) The **status** is *Uploaded*	* Step 1: (Message: **GetLog.req**) The **log.remoteLocation** is *<Configured log location>* The **logType** is *SecurityLog*

Test case name	Get Security Log	
Expected result(s) / behaviour	n/a	n/a

3.21.3. Secure firmware update

Secure Firmware Update

Table 187. Test Case Id: TC_080_CSMS

Test case name	Secure Firmware Update	
Test case Id	TC_080_CSMS	
Description	The firmware of a Charge Point is updated in a secure way.	
Purpose	To check whether Central System can trigger a Charge Point to update its firmware in a secure way.	
Prerequisite(s)	- The Central System supports the Firmware Management feature profile AND - The Central System supports a security profile.	
Prerequisite(s)	n/a	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point sends a **SignedUpdateFirmware.conf**	**1.** The Central System sends a **SignedUpdateFirmware.req**
	3. The Charge Point sends a **SignedFirmwareStatusNotification.req**	**4.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **SignedFirmwareStatusNotification.req**	**6.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
	[The Charge Point has verified the signature] **7.** The Charge Point sends a **SignedFirmwareStatusNotification.req**	**8.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
	[Before installing firmware the Charge Point MAY set all connectors to Unavailable. If the Charge Point supports installation of firmware during a charging session, the Charge Point MAY install the firmware after only setting all other connectors to Unavailable.] [The Charge Point starts installing the firmware] **9.** The Charge Point sends a **SignedFirmwareStatusNotification.req**	**10.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
	11. The Charge Point sends a **SignedFirmwareStatusNotification.req**	**12.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
	13. The Charge Point sends a **BootNotification.req**	**14.** The Central System responds with a **BootNotification.conf**
	15. The Charge Point sends a **SecurityEventNotification.req**	**16.** The Central System responds with a **SecurityEventNotification.conf**
	17. The Charge Point sends a **StatusNotification.req**	**18.** The Central System responds with a **StatusNotification.conf**
	[The Charge Point has finished installing the firmware] **19.** The Charge Point sends a **SignedFirmwareStatusNotification.req**	**20.** The Central System responds with a **SignedFirmwareStatusNotification.conf**

Test case name	**Secure Firmware Update**	
Tool validation(s)	* Step 3: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloading* * After step 2 and before step 9: Message: **StatusNotification.req** The **status** is *Unavailable* * Step 5: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *SignatureVerified* * Step 9: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Installing* * Step 11: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *InstallRebooting* * Step 15: (Message **SecurityEventNotification.req**) **type** *FirmwareUpdated* * Step 17: (Message: **StatusNotification.req**) The **status** is *Available* * Step 19: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Installed* * Step 13 / 15 / 17 / 19: The messages can be in a different order.	* Step 1: (Message: **SignedUpdateFirmware.req**) **firmware.location** is *<Configured Firmware Download URL>* **firmware.signature** is *<Configured signature>* **firmware.signingCertificate** is *<Configured signingCertificate>* After step 2 and before step 9: the CS responds to the **StatusNotification.req** with a **StatusNotification.conf**
Expected result(s) / behaviour	The Charge Point handles the firmware update correctly and is Available after the update.	The Central System receives and responds to the FirmwareStatusNotification messages.

Secure Firmware Update - Invalid Signature

Table 188. Test Case Id: TC_081_CSMS

Test case name	**Secure Firmware Update - Invalid Signature**
Test case Id	TC_081_CSMS
Description	The Charge Point validates the Signature and deems it invalid.
Purpose	To check whether the Central System is able to handle messages from a Charge Point when it reports that the signature is invalid.
Prerequisite(s)	- The Central System supports the Firmware Management feature profile AND - The Central System supports a security profile.
Before	**Configuration State(s):** n/a
	Memory State(s): n/a
	Reusable State(s): n/a

Test case name	**Secure Firmware Update - Invalid Signature**	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	2. The Charge Point sends a **SignedUpdateFirmware.conf**	**1.** The Central System sends a **SignedUpdateFirmware.req**
	[The Charge Point starts downloading the firmware] **3.** The Charge Point sends a **SignedFirmwareStatusNotification.req**	**4.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
	[The Charge Point has finished downloading the firmware] **5.** The Charge Point sends a **SignedFirmwareStatusNotification.req**	**6.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
	[The Charge Point verifies the signature and deems it invalid] **7.** The Charge Point sends a **SignedFirmwareStatusNotification.req**	**8.** The Central System responds with a **SignedFirmwareStatusNotification.conf**
Tool validation(s)	* Step 3: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloading* * Step 5: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *Downloaded* * Step 7: (Message: **SignedFirmwareStatusNotification.req**) The **status** is *InvalidSignature*	* Step 1: (Message: **SignedUpdateFirmware.req**) The **firmware.location** is *<Firmware Download URL from test data>* The **firmware.signature** is *<An invalid signature.>*
Expected result(s) / behaviour	The Charge Point rejects the firmware, because of an invalid signature.	The Central System receives and responds to the FirmwareStatusNotification messages.

Basic Authentication - Valid username/password combination

Table 189. Test Case Id: TC_085_CSMS

Test case name	**Basic Authentication - Valid username/password combination**	
Test case Id	TC_085_CSMS	
Description	The Charge Point uses Basic authentication to authenticate itself to the Central System, when using security profile 1 or 2.	
Purpose	To verify whether the Central System is able to validate the (valid) Basic authentication credentials provided by the Charge Point at the connection request.	
Prerequisite(s)	The Central System supports security profile 1 and/or 2.	
Before (Preparations)	**Configuration State:** N/a	
	Memory State: N/a	
	Reusable State(s): The OCTT closes the connection.	
Main (Test scenario)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends a HTTP upgrade request to the Central System	**2.** The Central System upgrades the connection to a WebSocket connection.
	3. The Charge Point sends a **BootNotification.req**	**4.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **5.** The Charge Point sends a **StatusNotification.req**	**6.** The Central System responds with a **StatusNotification.conf**
Tool validations	N/a	
	Post scenario validations: N/a	

TLS - server-side certificate - Valid certificate

Table 190. Test Case Id: TC_086_CSMS

Test case name	**TLS - server-side certificate - Valid certificate**	
Test case Id	TC_086_CSMS	
Description	The Central System uses a server-side certificate to identify itself to the Charge Point, when using security profile 2 or 3.	
Purpose	To verify whether the Central System is able to provide a valid server certificate and setup a secured WebSocket connection.	
Prerequisite(s)	The Central System supports security profile 2 and/or 3.	
Before (Preparations)	**Configuration State:** N/a	
	Memory State: N/a	
	Reusable State(s): The OCTT closes the connection.	
Main (Test scenario)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point initiates a TLS handshake and sends a Client Hello to the Central System.	**2.** The Central System responds with a Server Hello With the <Configured server certificate>
	3. The Charge Point performs the following actions: Send client certificate Client Key Exchange Certificate verify Change Cipher Spec Finished Note(s): *- The client certificate is only sent when the Central System uses security profile 3.*	**4.** The Central System performs the following actions: Change Cipher Spec Finished
	5. The Charge Point sends a HTTP upgrade request to the Central System Note(s): *- The HTTP request only contains a username/password combination when the Central System uses security profile 2.*	**6.** The Central System upgrades the connection to a (secured) WebSocket connection.
	7. The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**

Test case name	**TLS - server-side certificate - Valid certificate**
Tool validations	* Step 2: The OCTT validates the following before finishing the TLS handshake: - The Central System must use TLS version 1.2 or above At least the following set of cipher suites must be supported: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 AND TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 AND TLS_RSA_WITH_AES_128_GCM_SHA256 AND TLS_RSA_WITH_AES_256_GCM_SHA384 - When using RSA or DSA the key must be at least 2048 bits long. and when using elliptic curve cryptography the key must be at least 224 bits long. - The received server side certificate must be transmitted in the X.509 format encoded in Privacy-Enhanced Mail (PEM) format. - The certificate must include a serial number. - The subject field of the certificate must contain a commonName RDN which consists of the FQDN of the endpoint of the server. *NOTE: If one of the above validations fails, the OCTT can still proceed with the next steps of the testcase (if it is able to), but the testcase will FAIL and the OCTT reports why it failed.*
	Post scenario validations: N/a

TLS - Client-side certificate - valid certificate

Table 191. Test Case Id: TC_087_CSMS

Test case name	**TLS - Client-side certificate - valid certificate**
Test case Id	TC_087_CSMS
Description	The Charge Point uses a client-side certificate to identify itself to the Central System, when using security profile 3.
Purpose	To verify whether the Central System is able to receive a client certificate provided by a Charge Point and setup a secured WebSocket connection.
Prerequisite(s)	The Central System supports security profile 3.
Before (Preparations)	**Configuration State:** N/a
	Memory State: N/a
	Reusable State(s): The OCTT closes the connection.

Test case name	**TLS - Client-side certificate - valid certificate**	
Main (Test scenario)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point initiates a TLS handshake and sends a Client Hello to the Central System.	**2.** The Central System responds with a Server Hello With the <Configured server certificate>
	3. The Charge Point performs the following actions: Send client certificate Client Key Exchange Certificate verify Change Cipher Spec Finished	**4.** The Central System performs the following actions: Change Cipher Spec Finished
	5. The Charge Point sends a HTTP upgrade request to the Central System	**6.** The Central System upgrades the connection to a (secured) WebSocket connection.
	7. The Charge Point sends a **BootNotification.req**	**8.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0.] **9.** The Charge Point sends a **StatusNotification.req**	**10.** The Central System responds with a **StatusNotification.conf**
Tool validations	* Step 3: The OCTT validates the following before finishing the TLS handshake: - The Central System must use TLS version 1.2 or above At least the following set of cipher suites must be supported: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 AND TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 AND TLS_RSA_WITH_AES_128_GCM_SHA256 AND TLS_RSA_WITH_AES_256_GCM_SHA384	
	Post scenario validations: N/a	

3.22. Reusable states

Table 192. Reusable state: Booted

State	**Booted**	
Description	This state will simulate that the Charge Point is booting up.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends a **BootNotification.req** - **chargePointVendor** is *<Configured Vendor Name>* - **chargePointModel** is *<Configured Model>*	**2.** The Central System responds with a **BootNotification.conf**
	[Send per connector and connectorId=0] **3.** The Charge Point sends a **StatusNotification.req** - **status** is *Available*	**4.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 2: (Message: **BootNotification.conf**) - **status** should be *Accepted*	

State	**Booted**
Expected result(s) / behaviour	**State** is *Booted*

Table 193. Reusable state: Authorized

State	**Authorized**	
Description	This state will simulate that the EV Driver is locally authorizing to start a transaction on the simulated Charge Point.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends an **Authorize.req** - **idTag** is *<Configured Valid IdTag>*	**2.** The Central System responds with an **Authorize.conf**
Tool validation(s)	* Step 2: (Message: **Authorize.conf**) - **idTagInfo.status** should be *Accepted*	
Expected result(s) / behaviour	**State** is *Authorized*	

Table 194. Reusable state: Charging

State	**Charging**	
Description	This state will simulate that the Charge Point starts a transaction.	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): - *Authorized*	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	1. The Charge Point sends a **StatusNotification.req** - **status** is *Preparing* - **connectorId** is *<Configured ConnectorId>*	**2.** The Central System responds with a **StatusNotification.conf**
	3. The Charge Point sends a **StartTransaction.req** - **idTag** is *<Configured Valid IdTag>* - **connectorId** is *<Configured ConnectorId>*	**4.** The Central System responds with a **StartTransaction.conf**
	5. The Charge Point sends a **StatusNotification.req** - **status** is *Charging* - **connectorId** is *<Configured ConnectorId>*	**6.** The Central System responds with a **StatusNotification.conf**
Tool validation(s)	* Step 4: (Message: **StartTransaction.conf**) - **idTagInfo.status** should be *Accepted*	
Expected result(s) / behaviour	**State** is *Charging*	

Table 195. Reusable state: InstalledCertificatesReceived

State	**InstalledCertificatesReceived**
Description	This state will simulate that the CPO requests the installed root certificates on the Charge Point.

State	**InstalledCertificatesReceived**	
Before	**Configuration State(s):** n/a	
	Memory State(s): n/a	
	Reusable State(s): n/a	
Scenario Detail(s)	**Charge Point (Tool)**	**Central System (SUT)**
	Manual Action: *Request installed root certificates*	
	2. The Charge Point responds with a **GetInstalledCertificateIds.conf** - **certificateHashData** is *<Calculated hash data>*	**1.** The Central System sends a **GetInstalledCertificateIds.req**
Tool validation(s)	* Step 1: (Message: **GetInstalledCertificateIds.req**) - **certificateType** should be *<Expected certificateType>*	
Expected result(s) / behaviour	**State** is *InstalledCertificatesReceived*	