INTERNATIONAL STANDARD

ISO 15118-5

First edition
2018-02

Corrected version
2018-05

Road vehicles — Vehicle to grid communication interface —

Part 5: Physical layer and data link layer conformance test

Véhicules routiers — Interface de communication entre véhicule et réseau électrique —

Partie 5: Essai de conformité relatif à la couche physique et à la couche liaison de données

Reference number
ISO 15118-5:2018(E)

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html

This document was prepared jointly by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 31, *Data communication*, and Technical Committee IEC/TC 69, *Electric road vehicles and electric industrial trucks*. The draft was circulated for voting to the national bodies of both ISO and IEC.

A list of all parts in the ISO 15118 series can be found on the ISO website.

This corrected version of ISO 18541-6:2018 incorporates the following corrections:

— the foreword has been revised to indicate joint development with IEC/TC 69, *Electric road vehicles and electric industrial trucks*.

Introduction

The first two parts of ISO 15118 describe the use cases and the technical specification of the Vehicle-to-Grid Communication Interface which is intended for the optimized use of energy resources so that electric road vehicles can recharge in the most economic or most energy efficient way. It is furthermore required to develop efficient and convenient billing systems in order to cover micro-payments resulting from charging processes. The necessary communication channel may serve in the future to contribute to the stabilization of the electrical grid, as well as to support additional information services required to operate electric vehicles efficiently and economically.

Resulting from the physical and data link layer requirements defined in the third part of the standard, a corresponding set of test cases are required in order to verify conformance of implementations. This document therefore defines a conformance test suite for the physical and data link layer protocols in order to derive a common and agreed basis for conformance tests. The resulting test suite is a necessary prerequisite for downstream interoperability tests. Since interoperability furthermore involves the actual application logic of an implementation, those tests are beyond the scope of this document. Hence this document focuses on the interface aspects and the corresponding requirements given in part three only.

Road vehicles — Vehicle to grid communication interface — Part 5: Physical and data link layer conformance tests

1 Scope

This document specifies conformance tests in the form of an Abstract Test Suite (ATS) for a System Under Test (SUT) implementing an Electric Vehicle or Supply Equipment Communication Controller (EVCC or SECC) with support for PLC-based High Level Communication (HLC) and Basic Signaling according to ISO 15118-3. These conformance tests specify the testing of capabilities and behaviors of an SUT, as well as checking what is observed against the conformance requirements specified in ISO 15118-3 and against what the implementer states the SUT implementation's capabilities are.

The capability tests within the ATS check that the observable capabilities of the SUT are in accordance with the static conformance requirements defined in ISO 15118-3. The behavior tests of the ATS examine an implementation as thoroughly as is practical over the full range of dynamic conformance requirements defined in ISO 15118-3 and within the capabilities of the SUT (see NOTE 1).

A test architecture is described in correspondence to the ATS. The conformance test cases in this part of the standard are described leveraging this test architecture and are specified in TTCN-3 Core Language for the ISO/OSI Physical and Data Link Layers (Layers 1 and 2). The conformance test cases for the ISO/OSI Network Layer (Layer 3) and above are described in ISO 15118-4.

In terms of coverage, this document only covers normative sections and requirements in ISO 15118-3. This document can additionally include specific tests for requirements of referenced standards (e.g. IEEE, or industry consortia standards) as long as they are relevant in terms of conformance for implementations according to ISO 15118-3. However, it is explicitly not intended to widen the scope of this conformance specification to such external standards, if it is not technically necessary for the purpose of conformance testing for ISO 15118-3. Furthermore, the conformance tests specified in this document do not include the assessment of performance nor robustness or reliability of an implementation. They cannot provide judgments on the physical realization of abstract service primitives, how a system is implemented, how it provides any requested service, nor the environment of the protocol implementation. Furthermore, the test cases defined in this document only consider the communication protocol and the system's behavior defined ISO 15118-3. Power flow between the EVSE and the EV is not considered.

NOTE 1 Practical limitations make it impossible to define an exhaustive test suite, and economic considerations can restrict testing even further. Hence, the purpose of this document is to increase the probability that different implementations are able to interwork. This is achieved by verifying them by means of a protocol test suite, thereby increasing the confidence that each implementation conforms to the protocol specification. However, the specified protocol test suite cannot guarantee conformance to the specification since it detects errors rather than their absence. Thus conformance to a test suite alone cannot guarantee interworking. What it does do is give confidence that an implementation has the required capabilities and that its behavior conforms consistently in representative instances of communication.

NOTE 2 This document has some interdependencies to the conformance tests defined in ISO 15118-4 which result from ISO/OSI cross layer dependencies in the underlying protocol specification (e.g. for sleep mode)

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 61851-1:2017, *Electric vehicle conductive charging system — Part 1: General requirements (Ed 3.0, 2017)*

ISO 15118-1:2013, *Road vehicles — Vehicle to grid communication interface — Part 1: General information and use-case definition*

ISO 15118-2:2014, *Road vehicles — Vehicle-to-Grid Communication Interface — Part 2: Network and application protocol requirements*

ISO 15118-3:2015, *Road vehicles — Vehicle to grid communication interface — Part 3: Physical and data link layer requirements*

ETSI ES 201 873-5 V4.6.1, *TTCN-3: TTCN-3 Runtime Interface (June 2014)*

ETSI ES 201 873-6 V4.6.1, *TTCN-3: TTCN-3 Control Interface (June 2014)*

HomePlug Green PHY Specification, release version 1.1.1, July 4, 2013

NOTE 1 Even though ISO 15118-3:2015, which is the baseline for this conformance test document, explicitly references IEC 61851-1:2011, this document references IEC 61851-1:2017 because of applicability on the market.

3 Terms and definitions

For the purpose of this document, the terms and definitions given in ISO 15118-1, ISO 15118-2, ISO 15118-3 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at https://www.iso.org/obp

3.1
abstract test case
complete and independent specification of the actions required to achieve a specific test purpose

Note 1 to entry: This specification is defined at the level of abstraction of a particular Abstract Test Method, starting in a stable testing state and ending in a stable testing state and may involve one or more consecutive or concurrent connections.

Note 2 to entry: The specification should be complete in the sense that it is sufficient to enable a test verdict to be assigned unambiguously to each potentially observable test outcome (i.e. sequence of test events).

Note 3 to entry: The specification should be independent in the sense that it should be possible to execute the derived executable test case in isolation from other such test cases (i.e. the specification should always include the possibility of starting and finishing in the “idle” state).

Note 4 to entry: Compare with ITU-T X.290.

3.2
abstract test suite
ATS
test suite composed of abstract test cases

Note 1 to entry: Compare with ITU-T X.290.

3.3
black box testing
method of testing that examines the behavior of an SUT without considering the internal implementation and structure of the SUT, thus relying on the SUT's open interface for testing

3.4
conformance requirements
conformance of a real system consisting of conformance to each requirement and conformance to the set

Note 1 to entry: Set of interrelated requirements which together define the behavior of the system and its communication. Conformance of a real system will, therefore, be expressed at two levels, conformance to each individual requirement and conformance to the set. Applicable ISO 15118-4 conformance tests include requirements and transfer syntax requirements as far as they can be validated by black box testing.

Note 2 to entry: See also *static conformance requirements* (3.20) and *dynamic conformance requirements* (3.6).

3.5
conforming implementation
IUT which satisfies both static and dynamic conformance requirements, consistent with the capabilities stated in the PICS(s)

Note 1 to entry: Compare with ITU-T X.290.

3.6
dynamic conformance requirements
one of the requirements which specifies what observable behavior is permitted by the relevant specification(s) in instances of communication

Note 1 to entry: The requirements for this conformance specification are defined in ISO 15118-3.

Note 2 to entry: Compare with ITU-T X.290.

3.7
executable test case
realization of an abstract test case

Note 1 to entry: Compare with ITU-T X.290.

3.8
expected behavior
exact response of the SUT according to the underlying protocol specification to the stimulus defined in the test behavior

3.9
implementation conformance statement
ICS
statement made by the supplier of an implementation or system claimed to conform to a given specification, stating which capabilities have been implemented

Note 1 to entry: The given specification for this conformance specification is ISO 15118-3.

Note 2 to entry: Compare with ITU-T X.290.

3.10

implementation extra information for testing
IXIT
statement made by a supplier or implementer of an IUT which contains or references all of the information (in addition to that given in the ICS) related to the IUT and its testing environment, which will enable the test laboratory to run an appropriate test suite against the IUT

Note 1 to entry: Compare with ITU-T X.290.

3.11
implementation under test
IUT
implementation of one or more OSI protocols in an adjacent user/provider relationship, being that part of a real open system which is to be studied by testing

Note 1 to entry: Compare with ITU-T X.290.

3.12
main test component
MTC
single test component in a test component configuration responsible for creating and controlling *parallel test components* and computing and assigning the test verdict

Note 1 to entry: Compare with ITU-T X.292.

3.13
parallel test component
PTC
test component created by the main test component

Note 1 to entry: Compare with ITU-T X.292.

3.14
post-condition
test steps needed to define the path from the end of the *test behavior* up to the finishing stable state for the test case

Note 1 to entry: See also *test behavior* (3.23).

3.15
pre-condition
test steps needed to define the path from the starting stable state of the test case up to the initial state from which the *test bevavior* will start

Note 1 to entry: See also *test behavior* (3.23).

3.16
protocol implementation conformance statements
PICS
ICS for an implementation or system claimed to conform to a given protocol specification

Note 1 to entry: The given protocol specification for this conformance specification is ISO 15118-3.

Note 2 to entry: Compare with ITU-T X.290.

3.17
protocol implementation extra information for testing
PIXIT

IXIT related to testing for conformance to a given protocol specification

Note 1 to entry: The given protocol specification for this conformance specification is ISO 15118-3.

Note 2 to entry: Compare with ITU-T X.290.

3.18
runtime environment
environment that describes the operating system and corresponding platform requirements of a system

EXAMPLE *Test system*.

3.19
semantically invalid test behavior
SemITB
test steps where the test system sends stimuli to the SUT that are semantically invalid according to the protocol requirements

Note 1 to entry: This type of test behavior is defined in this document and explicitly includes requirements which define the appropriate error handling of the SUT.

3.20
static conformance requirements
one of the requirements that specify the limitations on the combinations of implemented capabilities permitted in a real open system which is claimed to conform to the relevant specification(s)

Note 1 to entry: Compare with ITU-T X.290.

3.21
system under test
SUT
real open system in which the IUT resides

Note 1 to entry: Compare with ITU-T X.290.

3.22
syntactically invalid test behavior
SynITB
test steps where the test system sends stimuli to the SUT that are syntactically invalid according to the protocol requirements

Note 1 to entry: This type of test behavior is not defined in this conformance standard, see codec requirements.

3.23
test behavior
set of test steps (test body) which are essential in order to achieve the test purpose and assign verdicts to the possible outcomes

3.24
test execution
interpretation or execution of an abstract test suite

Note 1 to entry: Conceptually, the TE can be decomposed into three interacting entities: an Executable Test Suite (ETS), a Test Framework (TFW) and an optional internal Encoding/Decoding System (EDS) entity.

Note 2 to entry: See also ETSI ES 201 873-5 V4.6.1.

3.25
test framework
TFW
entity to perform all actions of test cases or functions

Note 1 to entry: The Test Framework interacts with the Test Management (TM), SUT Adaptor (SA) and Platform Adaptor (PA) entities via Test Control Interface (TCI) and Test Runtime Interface (TRI) and additionally manages the Executable Test Suite (ETS) and Encoding/Decoding System (EDS) entities. It initializes adaptors as well as ETS and EDS entities. This entity performs all the actions necessary to properly start the execution of a test case or function with parameters in the ETS entity. It queries the TM entity for module parameter values required by the ETS and sends logging information to it. It also collects and resolves associated verdicts returned by the ETS entity.

Note 2 to entry: See also ETSI ES 201 873-5 V4.6.1.

Note 3 to entry: In this document, the Test Framework TTCN-3 Runtime System (T3RTS) is used to explain a Test Framework functionality.

3.26
test purpose
prose description of a well-defined objective of testing, focusing on a single conformance requirement or a set of related conformance requirements as specified in the appropriate OSI specification

EXAMPLE Verifying the support of a specific value of a specific parameter.

Note 1 to entry: Compare with ITU-T X.290.

3.27
test system
real system combining the test framework, abstract test suite, test execution and adapters as well as codecs

Note 1 to entry: Typically also containing a common runtime environment based on an operating system.

3.28
test control interface
TCI
four interfaces that define the interaction of the TTCN-3 Executable with the test management, the coding and decoding, the test component handling and the logging in a test system

Note 1 to entry: Compare with ITU-T X.290.

Note 2 to entry: Compare with ETSI ES 201 873-6 V4.6.1.

3.29
test runtime interface
TRI
two interfaces that define the interaction of the TTCN-3 Executable between the SUT and the Platform Adapter (PA) and the System Adapter (SA) in a test system

Note 1 to entry: Compare with ETSI ES 201 873-5 V4.6.1.

3.30
test system interface
TSI
test component that provides a mapping of the ports available in the (abstract) TTCN-3 test system to those offered by a real test system

Note 1 to entry: Compare with ETSI ES 201 873-6 V4.6.1.

3.31
valid test behavior
VTB
test steps where the test system sends stimuli to the SUT that are valid (syntactically and semantically) according to the protocol requirements

Note 1 to entry: This type of test behavior is defined in this conformance document.

Note 2 to entry: The protocol requirements for this conformance specification are defined in ISO 15118-3.

3.32
verdict
test verdict
statement of "pass", "fail" or "inconclusive", as specified in an abstract test case, concerning conformance of an IUT with respect to that test case when it is executed

Note 1 to entry: Compare with ITU-T X.290.

4 Symbols (and abbreviated terms)

For the purposes of this document, the following abbreviations apply:

AC	Alternating Current
ATS	Abstract Test Suite
CPL	Control Pilot Line
DC	Direct Current
EIM	External Identification Means
ETSI	European Telecommunications Standards Institute
EV	Electric Vehicle
EVCC	Electric Vehicle Communication Controller
EVSE	Electric Vehicle Supply Equipment
HAL	Hardware Abstraction Layer
HPGP	HomePlug GreenPHY
ITB	Invalid Test Behavior
MME	Management Message Entry
MTC	Main Test Component
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
PLC	Power Line Communication
PnC	Plug and Charge
PTC	Parallel Test Component
PWM	Pulse Width Modulation
SECC	Supply Equipment Communication Controller
SLAC	Signal Level Attenuation Characterization

SUT	System Under Test
TC	Test Case
TCI	TTCN-3 Control Interface
TCI-CD	TCI-Coding and Decoding
TE	Test Execution
TFW	Test Framework
TP	Test Purpose
TRI	TTCN-3 Runtime Interface
TSI	TTCN-3 System Interface
TSS	Test Suite Structure
TTCN-3	Testing and Test Control Notation version 3
V2G	Vehicle-to-Grid
VTB	Valid Test Behavior

5 Conventions

5.1 Requirement structure

This document uses unique number identifiers for each individual requirement. This requirement structure allows for easier requirement tracking and management. The following format is used throughout this document:

"[V2G"Y"-"XXX"]" requirement text

Where:

— "V2G" represents the ISO 15118 set of standards;

— Y represents the document part of the ISO 15118 document set, for this document Y = 5;

— XXX represents the individual requirement number; and

— "requirement text" includes the actual text of the requirement.

5.2 Test system description

TTCN-3 is used in this document to define/specify the test system and the test cases inside the test suite. However, using TTCN-3 is not mandatory for the implementation of conformance tests for ISO 15118-3.

[V2G5-001] The implementers of conformance tests shall verify that the test objectives implemented in their environment are identical to those described in this document.

6 Test architecture reference model

6.1 General information

Figure 1 provides an overview on the test architecture for this document. The following subclauses define the interface requirements marked black in Figure 1 for platform and SUT adapters (see 6.2, 6.3) as well as the codecs (see 6.4). The Test Suite is defined in detail in the remainder of this document.

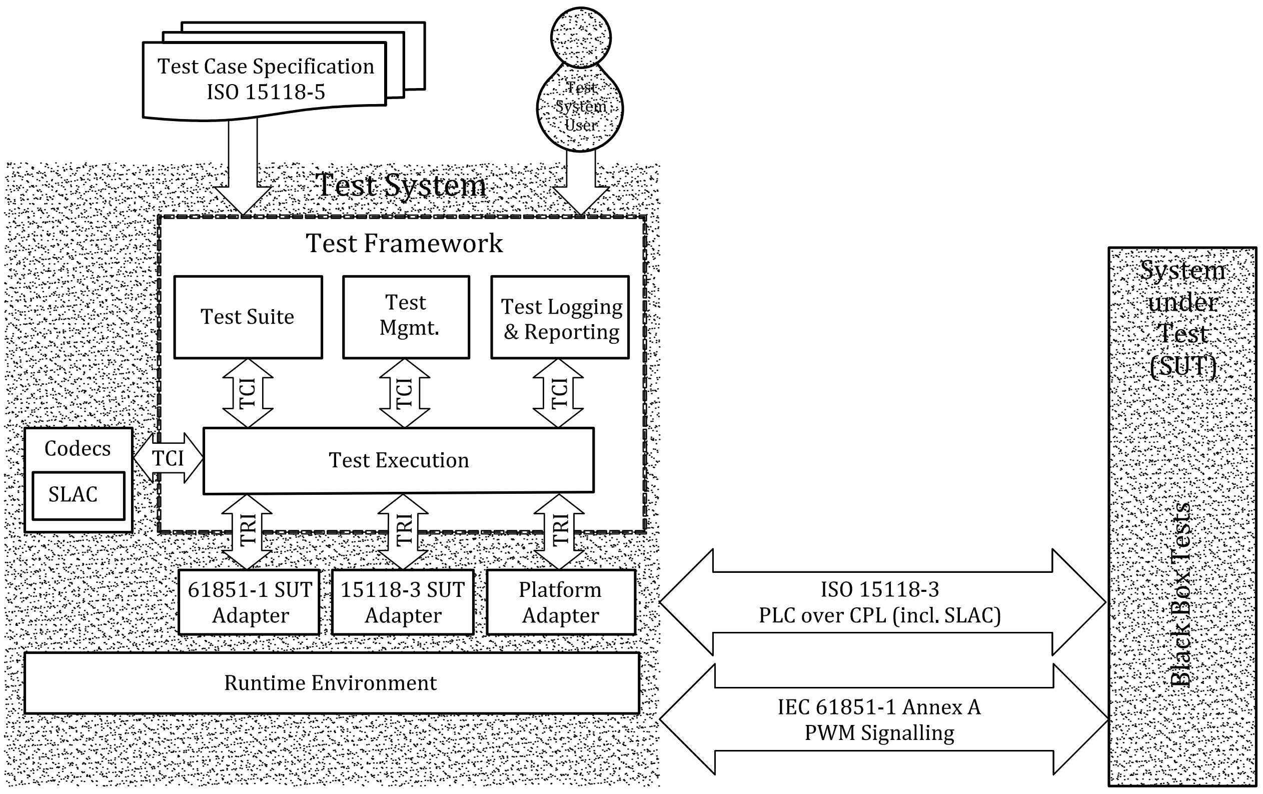

Figure 1 — Test architecture reference model for testing of ISO 15118-3

6.2 Platform adapter interface

The platform adapter within the test system is responsible for timers and external functions. Besides means for timers, which are typically provided as part of the test framework, no external functions are defined for ISO 15118 conformance testing.

[V2G5-002] The platform adapter of the test system shall implement the TriPlatformPA and the TriPlatformTE interfaces as defined in ETSI ES 201 873-5 V4.6.1, 6.5.3.

6.3 SUT adapter interfaces

The SUT adapter within the test system adapts the TTCN-3 communication operations to the SUT based on an abstract test system interface and implements the real test system interface. It is responsible to propagate message requests and procedure based calls from the Test Execution (see Figure 1) to the SUT, and to notify the Test Execution of any received test events by appending them to its port queues.

[V2G5-003] The SUT adapters of the test system shall implement the TriCommunicationSA and the TriCommunicationTE interfaces as defined in ETSI ES 201 873-5 V4.6.1, 6.5.2.

NOTE 1 The actual implementation of these adapters is out of scope of this document.

[V2G5-004] The IEC 61851-1 SUT adapter shall adapt the HAL 61851-1 functions to a PWM Signal defined in 7.7.7.

[V2G5-005] The ISO 15118-3 SUT adapter shall send/receive the encoded SLAC request/response messages through raw Ethernet frames to/from the SUT as defined in ISO 15118-3:2015, A.6 and A.9.

NOTE 2 The IEC 61851-1 SUT adapter can need additional hardware for adapting the HAL 61851-1 functions to PWM Signalling according to IEC 61851-1:2017, Annex A.

6.4 Codecs

A codec is responsible for the external encoding and decoding of TTCN-3 values into bit strings suitable to be sent to the SUT. The Test Execution (TE) determines which codec shall be used and passes the TTCN-3 data to the appropriate encoder to obtain the encoded data. Received data is decoded in this entity by using the appropriate decoder, which translates the received data into TTCN-3 values cf. ETSI ES 201 873-6 V4.6.1 that can be matched against expected values or templates.

[V2G5-006] All codecs in this document shall implement the TCI-CD interface as defined in ETSI ES 201 873-6 V4.6.1, 7.3.2.

NOTE 1 For ISO 15118-3 conformance testing the SLAC codec (see Figure 1) is used to encode or decode messages consumable by the tester into bit strings consumable by the SUT.

NOTE 2 The exact implementation of the SLAC codec is out of scope of this document.

[V2G5-007] The SLAC codec shall encode TTCN-3 values as defined in G.3 into corresponding MME frames as defined in ISO 15118-3:2015, A.9 and HomePlug GreenPHY Spec. 1.1.1, July 4, 2013.

[V2G5-008] The SLAC codec shall decode MME frames as defined in G.3 into TTCN-3 values as defined in ISO 15118-3:2015, A.9 and HomePlug GreenPHY Spec. 1.1.1, July 4, 2013.

7 Test suite conventions

7.1 General information

This clause defines all conventions that are relevant for conformance tests of SUTs implementing ISO 15118-3.

7.2 Test suite structure (TSS)

A test suite is a complete set of test cases, possibly combined into groups or modules (e.g. for use cases or domains like AC, DC charging), that are necessary to perform conformance testing for a given SUT.

In each test case, the SUT is stimulated with specific inputs and the reactions are observed and evaluated. Depending on the test purpose different pre-conditions and post-conditions shall be considered for the formulation of the test behavior. The pre-conditions, post-conditions as well as test behaviors are encapsulated into individual functions and stored within separate modules. Thus, a complete test case is composed by the actual test behavior enveloped by pre- and post-conditions (see 7.7.5 for details). The corresponding grouping of functions can therefore be assigned to the lowest abstract hierarchical level (see Figure 2). The test cases are defined on the second level.

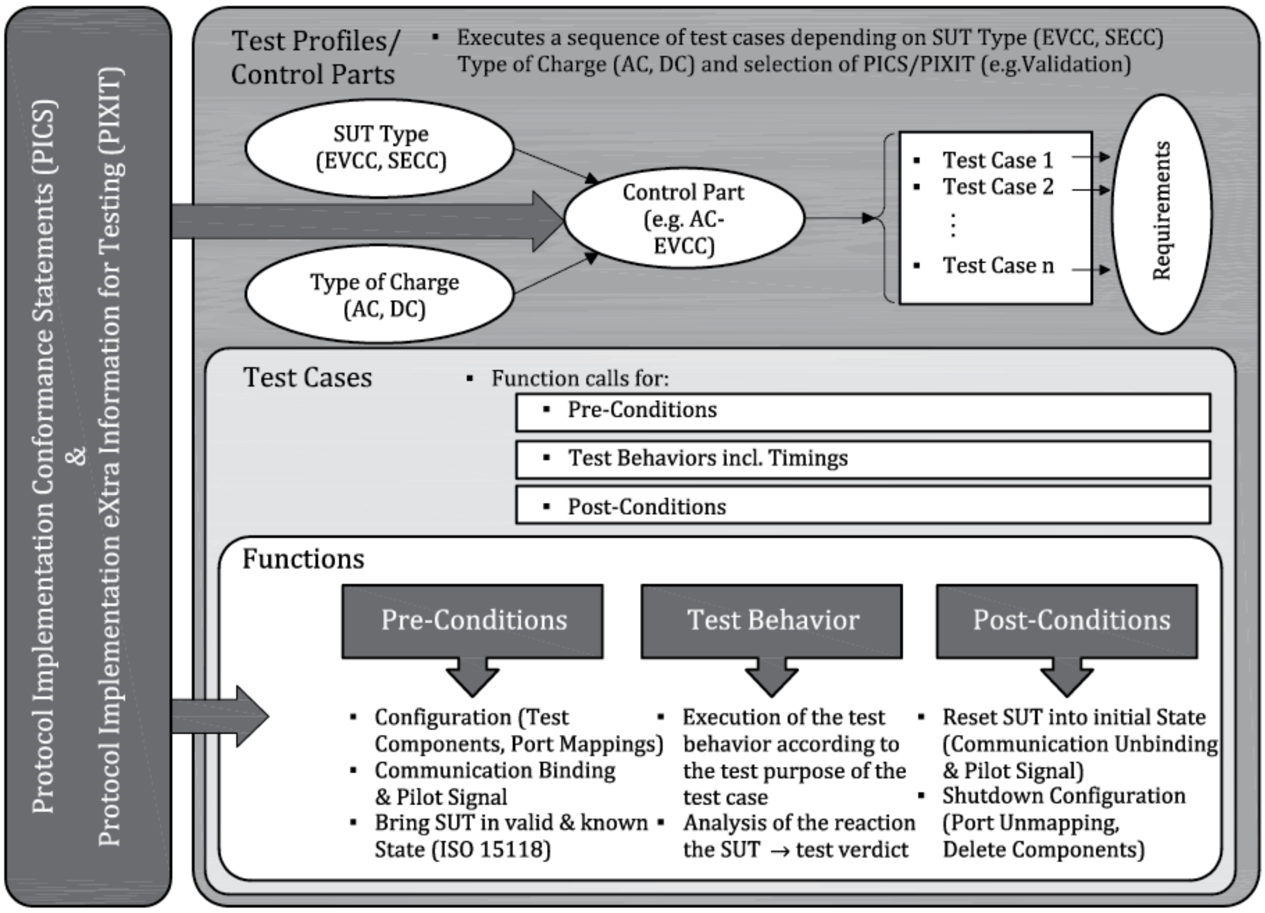
Protocol Implementation Conformance Statements (PICS)
&
Protocol Implementation eXtra Information for Testing (PIXIT)
Test Profiles/ Control Parts
Executes a sequence of test cases depending on SUT Type (EVCC, SECC) Type of Charge (AC, DC) and selection of PICS/PIXIT (e.g.Validation)
SUT Type (EVCC, SECC)
Type of Charge (AC, DC)
Control Part (e.g. AC-EVCC)
Test Case 1
Test Case 2
Test Case n
Requirements
Test Cases
Function calls for:
Pre-Conditions
Test Behaviors incl. Timings
Post-Conditions
Functions
Pre-Conditions
Test Behavior
Post-Conditions
Configuration (Test Components, Port Mappings)
Communication Binding & Pilot Signal
Bring SUT in valid & known State (ISO 15118)
Execution of the test behavior according to the test purpose of the test case
Analysis of the reaction the SUT → test verdict
Reset SUT into initial State (Communication Unbinding & Pilot Signal)
Shutdown Configuration (Port Unmapping, Delete Components)

Figure 2 — General overview of the Test Suite Structure (TSS)

The test profile is a collection of self-contained test cases as well as PICS (see 7.3.3) and PIXIT (see 7.3.4) in order to represent a given use case. The selection is based on the use cases of the ISO 15118 standard and its corresponding requirements.

Hence the Test Suite Structure (TSS) is segmented into subgroups defined according to ISO 15118 use cases for conformance testing. Table 1 shows these subgroups, which are used for the organization of the test case specifications as well as for the test suite identifiers (see 7.4 for details).

Table 1 — Identifiers within the Test Suite Structure (TSS)

Identifiers	Values	Description
<sut>		System under Test
	EVCC	Electric Vehicle Communication Controller
	SECC	Supply Equipment Communication Controller
<dom>		Domain
	AC	AC specific behaviors
	DC	DC specific behaviors
	IN	Inductive specific behaviors
	CMN	Common behaviors
<ctx>	{fullname}	Context (e.g. name of message pattern signal name according to standard)

NOTE 1 The domain for inductive charging is not relevant for this document but was is introduced for future purposes.

Table 2 describes the Annexes derived for this document.

Table 2 — Test suite structure — Annexes description

Annexes	Description
Configuration Annex A	The configuration annex contains constant definitions timers, PICS and PIXIT definitions. The configuration in Annex A shall be applied.
Control Part Annex B	The control part annex describes the execution order of modelled test cases (test groups). The EVCC control part subclauses contain the test groups if the SUT is an EVCC. The SECC control part subclauses contain the test groups if the SUT is an SECC. The control part in Annex B shall be followed.
Test Cases Annex C	The test case annexes contain the TTCN-3 test case files. The EVCC subclauses contain the test cases if the SUT is an EVCC. The SECC subclauses contain the test cases if the SUT is an SECC. The test cases in Annex C shall be considered.
Functions Annex D - E	The function annexes contain methods for supporting the test execution as well as the actual pre-, post-conditions and test behaviors of the ISO 15118 conformance tests. The EVCC subclauses contain the test behavior if the SUT is an EVCC. The SECC subclauses contain the test behavior if the SUT is an SECC. The Pre-condition subclause contains functions that are used for defining the pre-conditions of a test behavior. The Post-condition subclause contains functions that are used to establish the post-conditions of a test case. The Library Functions subclause contains utility functions used across the test suite for various purposes. The functions in Annex D and Annex E shall be considered.
Templates Annex F	The Templates subclause contains the TTCN-3 template files for matching a SUT's reaction on a stimulus against its expected behavior. The templates in Annex F shall be considered.
Data Structures Annex G	The Data Structures subclause contains data structures that are needed for testing of SLAC. The data structures in Annex G shall be considered.

7.3 Test profiles

This subclause defines test profiles for conformance with ISO 15118-3. A test profile consists of a test configuration as well as a selection and assignment of PICS/PIXIT. Depending on the test configuration a set of test components and ports are defined. The test profile furthermore includes a test group defining the set of relevant test cases and the sequence in which they are executed in order to perform a conformance test for a given use case.

7.3.1 Test configurations

The test configuration reflects various ISO 15118 scenarios. The main entities for the System Under Test (SUT) are:

— Electric Vehicle Communication Controller (EVCC); and

— Supply Equipment Communication Controller (SECC)

The combination of entities and additionally used test components are grouped by Test Configuration IDs (CF_Part_ID). Table 3 shows the test configurations for this document.

Table 3 — ISO 15118-5 test configurations

CF_Part_ID	SUT	Tester	PTCs
CF_05_001	SECC + PLC Bridge	EVCC + PLC Bridge	PTC1 = 61851
CF_05_002	EVCC + PLC Bridge	SECC + PLC Bridge	PTC1 = 61851

7.3.2 Components and ports

In correspondence to the identified set of relevant test configurations this sub clause defines test components which reflect the main entities needed for stimulation of the SUT with respect to ISO 15118. Ports are used to connect these components with each other and the SUT. Port types define which kind of messages can be sent or received by this port. All relevant components and ports are defined in Table 4 and Table 5 respectively.

Table 4 — Component definitions

Components	Description
SECC_Tester (MTC)	This component type is the main type for the tests of an SECC. A SLAC_Port, HAL_61851_Port and a HAL_61851_Internal_Port (see Table 5) are assigned to this component type.
EVCC_Tester (MTC)	This component type is the main type for the tests of an EVCC. A SLAC_Port, HAL_61851_Port and a HAL_61851_Internal_Port (see Table 5) are assigned to this component type.
HAL_61851_Listener (PTC)	The HAL_61851_Listener is responsible for watching the correct IEC 61851 behavior and is used in the form of a parallel test component (PTC). This component uses the HAL_61851_Port (see Table 5) to communicate with an IEC 61851-1 SUT Adapter and the HAL_61851_Internal_Port to communicate with the MTC.

Table 5 — Port type definitions

Port Type	Description
SLAC_Port	This port is used to send/receive SLAC messages defined in ISO 15118-3 to/from the EVCC/SECC.
HAL_61851_Port	This port is used to interact with an IEC 61851 signalling unit. The unit will set the corresponding signalling to the SUT (SECC/EVCC). The specification of the IEC 61851 signalling unit is out of scope of this standard.
HAL_61851_Internal_Port	This port is used to synchronize the status of the IEC 61851-1 PWM signal between the MTC (SECC_Tester or EVCC_Tester) and PTC (HAL_61851_Listener_Port).

These components and ports compose relevant test configurations for this document. Whether the type EVCC_Tester or SECC_Tester is to be used as MTC depends on the type of the SUT.

[V2G5-009] If the SUT is an EVCC, the MTC shall use the type EVCC_Tester.

[V2G5-010] If the SUT is an SECC, the MTC shall use the type SECC_Tester.

The HAL_61851_Listener combines all necessary listener functionality into one component, independent on whether the MTC represents an SECC_Tester or an EVCC_Tester. The MTC always contains a TTCN-3 test configuration and delimits the lifeline during test execution. Next to using ports for communication purposes, local timers, variables or constants may be assigned to components to store dynamic information during test case execution.

In addition to the MTC and PTC and their corresponding port type definitions, a test configuration also consists of respective Test System Interfaces (TSI). An abstract Test System Interface (TSI) is specified as a collection of ports. A TSI has no local timers, constants or variables. Only ports are assigned to it. During the test case execution test components ports can be mapped dynamically to the TSI ports to establish communication channel to the real test system interface. In the test configuration the TSI uses the type System_EVCC or System_SECC depending on the type of the SUT. If the SUT is an EVCC, the TSI uses the type System_EVCC. If the SUT is an SECC, the TSI uses the type System_SECC.

The test configuration is illustrated in Figure 3. The type of the V2G components and ports (EVCC or SECC) depends on the SUT type.

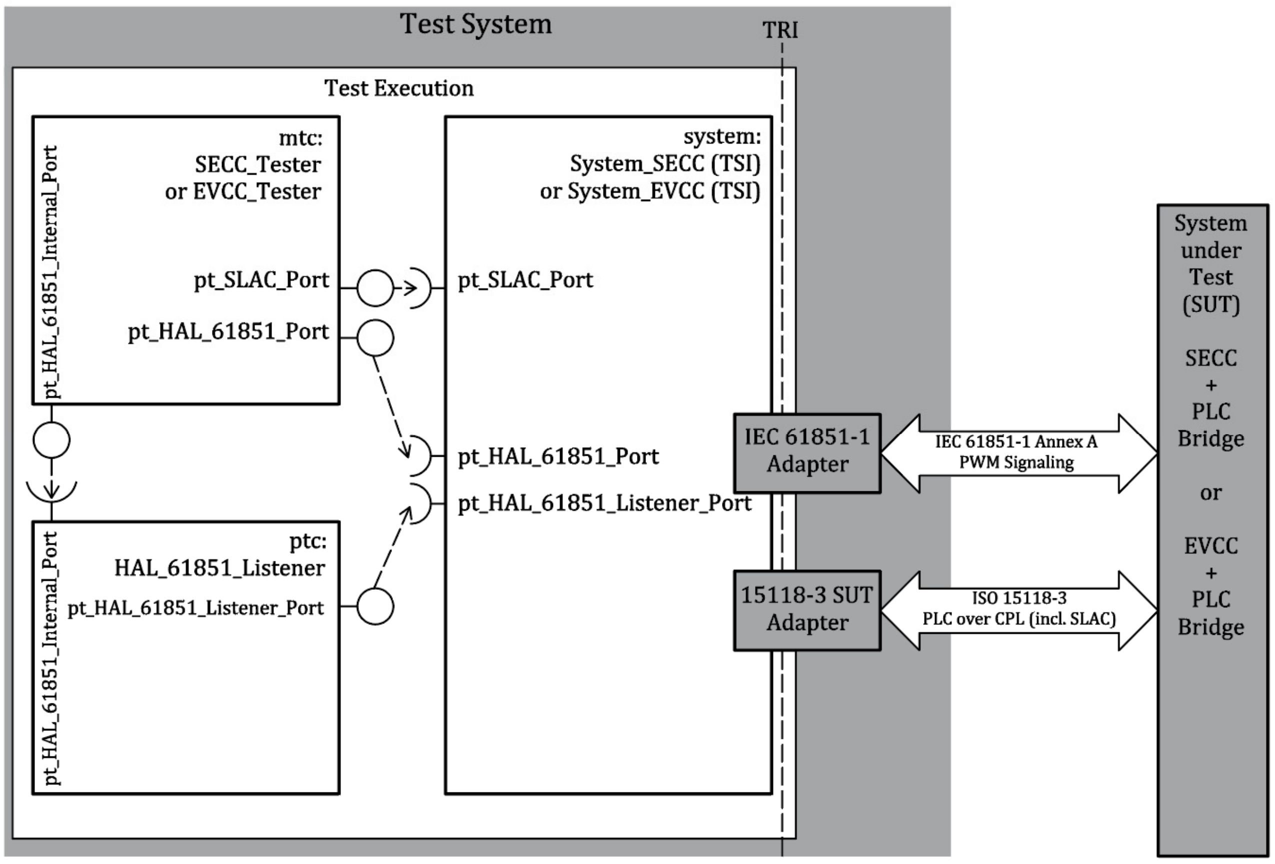

Figure 3 — ISO 15118-5 test configuration

As shown in Figure 3 the port mappings are defined statically as follows:

— The port pt_SLAC_Port of the TSI is always mapped to port pt_SLAC_Port of the MTC.

— The port pt_HAL_61851_Port of the TSI is always mapped to port pt_HAL_61851_Port of the MTC.

— The port pt_HAL_61851_Listener_Port of the TSI is always mapped to port pt_HAL_61851_Listener_Port of the v_HAL_61851_Listener. The pt_HAL_61851_Listener_Port is from the same type HAL_61851_Port as the pt_HAL_61851_Port.

— The internal port pt_HAL_61851_Internal_Port is not mapped to the TSI. This port is connected between the MTC and PTC.

7.3.3 Protocol implementation conformance statement (PICS) definition

To evaluate the conformance of a particular implementation, it is necessary to have a statement of the capabilities and options which have been implemented, and any features which have been omitted, so that the implementation can be tested for conformance against relevant requirements, and against those requirements only. Such a statement is called a Protocol Implementation Conformance Statement (PICS) cf. [ITU-T X.290].

All PICS defined in the ATS are summarized in Table 6, Table 7 and Table 8.

Table 6 — PICS for test system configurations CF_05_001 and CF_05_002 (SUT either SECC or EVCC)

PICS	Description
PICS_CMN_CMN_CombinedTesting	Indication for enabling combined testing including SLAC

	association and V2G messaging
PICS_CMN_CMN_ChargingMode	Indication for testing either AC or DC charging mode
PICS_CMN_CMN_IdentificationMode	Indication for testing either EIM or PnC identification mode
PICS_CMN_CMN_PlugType	Indication for testing either type1 or type2 plug type
PICS_CMN_AC_CableCapability	Indication for used AC cable type Choice: i) capability13A, ii) capability20A, iii) capability32A, iv) capability63A, v) capability70A
PICS_CMN_CMN_InitiateCmAmpMap	Indication for initiating the Amplitude map process
PICS_CMN_CMN_WakeUp	Indication for the sleep time within a paused session
PICS_CMN_CMN_SlacTimeouts	Indication for enabling Test Group: SLAC timeouts
PICS_CMN_CMN_InvalidSlacDataFieldsAndMessages	Indication for enabling Test Group: Invalid SLAC data fields and messages
PICS_CMN_CMN_InvalidStatesAndDutyCycles	Indication for enabling Test Group: Invalid states and duty cycles

Table 7 — PICS for test system configuration CF_05_001 (SUT equals SECC)

PICS	Description
PICS_SECC_CMN_Pause	Indication for enabling a charging pause
PICS_SECC_CMN_EIMDone	Indication for initiating EIM Authorization process Choice: i) beforePlugin, ii) afterPlugin, iii) duringSlac, iv) v2gAuthorization

Table 8 — PICS for test system configuration CF_05_002 (SUT equals EVCC)

PICS	Description
PICS_EVCC_CMN_PmaxSchedulewithZeroPow	Indication for enabling a PmaxScheduleList with a list element pMax = 0W (triggering pause) Choice: i) sleepWithoutCharge, ii) sleepAfterCharge, iii) none_

7.3.4 Protocol implementation extra information for testing (PIXIT) definition

In order to ensure testability of ISO 15118-3 requirements depending on a specific behavior of the SUT, the following set of PIXIT is defined in addition to PICS in this document. PIXIT are used to indicate to the SUT that a specific capability of the SUT is tested.

NOTE 1 Due to the black box test paradigm in this document, it is not defined how to ensure that a corresponding PIXIT is set on the SUT side for a given test case execution.

All PIXIT defined in the ATS are summarized in Table 9 - Table 11.

Table 9 — Selected PIXIT for test system configurations CF_05_001 and CF_05_002 (SUT either SECC or EVCC)

PIXIT	Description
PIXIT_CMN_CMN_CmAmpMap	Indication for explicit testing of Amplitude map process
PIXIT_CMN_CMN_WakeUp	Indication for the wake-up time (SUT) within a paused session

Table 10 — Selected PIXIT for test system configuration CF_05_001 (SUT equals SECC)

PIXIT	Description
PIXIT_SECC_CMN_CmValidate	Indication for explicit testing of SECC Validation process Choice: i) none_, ii) cmValidate
PIXIT_SECC_AC_InitialDutyCyle	Indication for starting with initial duty cycle of 5 % or 100 %
PIXIT_SECC_CMN_ArchitectureValidationNotRequired	Indication for SECC grid architecture which not requires a validation process
PIXIT_SECC_AC_ConnectionLossHandling	Indication for the implemented handling after connection loss (Option A: EVSE shall leave the logical network and shall switch to state E/F after connection loss, Option B: EVSE shall leave the logical network, shall stay in X2 state and waits for a new matching process)

Table 11 — Selected PIXIT for test system configuration CF_05_002 (SUT equals EVCC)

PIXIT	Description
PIXIT_EVCC_CMN_CmValidate	Indication for explicit testing of EVCC Validation process. If PIXIT_EVCC_CMN_CmValidate := cmValidate, test system sends a manipulated attenuation profile (attenuation was increased) to trigger a validation process Choice: i) cmValidate, ii) unknown, iii) none_
PIXIT_EVCC_CMN_FallbackValidationFailed	Indication for validation fallback handling if the SUT does not have implemented the validation process feature (failed) Choice: i) continue_, ii) skip, iii) terminate iv) unknown
PIXIT_EVCC_CMN_FallbackValidationNotRequired	Indication for validation fallback if the SECC grid architecture not requires a validation process (notRequired) Choice: i) continue_, ii) skip, iii) unknown
PIXIT_EVCC_CMN_ConcurrentValidation	Indication for handling if an SECC is occupied by another running validation process Choice: i) retry, ii) iterate, iii) unknown
PIXIT_EVCC_CMN_TTMatchingRepetitionConfig	Indication for the knowledge of the configuration parameter PIXIT_EVCC_CMN_TTMatchingRepetition and PIXIT_EVCC_CMN_TTMatchingRate
PIXIT_EVCC_CMN_TTMatchingRepetition	Predefined SUT value for the parameter TT_matching_repetition
PIXIT_EVCC_CMN_TTMatchingRate	Predefined SUT value for the parameter TT_matching_rate
PIXIT_EVCC_CMN_ValidationRetry	Predefined SUT value for time to wait for a retry of the validation process if a SECC is occupied by another running validation process
PIXIT_EVCC_CMN_Pause	Indication for enabling a charging pause Choice: i) pause, ii) unknown, iii) none_
PIXIT_EVCC_AC_ConnectionLossHandling	Indication for the implemented handling after connection loss (Option A: EV shall leave the logical network and shall wait for a new incoming matching trigger (control pilot X1 or X2 state), Option B: EV shall leave the logical network and shall restart the matching process after 'T_conn_resetup'.
PIXIT_EVCC_AC_TconnResetup	Predefined SUT value for the parameter T_conn_resetup

7.3.5 Test control

The control part calls the test cases with actual parameters and controls their execution. Program statements PICS/PIXIT are used to specify the selection and execution order of the test cases (see Annex B). For each use case all valid PICS/PIXIT configurations are summarized in Table 12 - Table 15.

Table 12 — SECC AC PICS/PIXIT configuration

Configuration	EIM	PnC

1	PIXIT_SECC_AC_InitialDutyCyle := dc5 **and**
	PICS_CMN_CMN_IdentificationMode := eIM **and**
	PICS_SECC_CMN_EIMDone := afterPlugin **and**
	PICS_CMN_CMN_ChargingMode := aC
		PIXIT_SECC_CMN_CmValidate == cmValidate := cmValidate **and**
		PICS_CMN_CMN_ChargingMode := aC

2	PIXIT_SECC_AC_InitialDutyCyle := dc100 **and**
	PICS_CMN_CMN_IdentificationMode := eIM **and**
	PICS_SECC_CMN_EIMDone := afterPlugin **and**
	PICS_CMN_CMN_ChargingMode := aC
		PIXIT_SECC_CMN_CmValidate := cmValidate **and**
		PIXIT_SECC_CMN_ArchitectureValidationNotRequired := true **and**
		PICS_CMN_CMN_ChargingMode := aC

3	PICS_CMN_CMN_IdentificationMode := eIM **and**
	PICS_SECC_CMN_EIMDone := beforePlugin **and**
	PICS_CMN_CMN_ChargingMode := aC
		PIXIT_SECC_CMN_CmValidate := none_ **and**
		PICS_CMN_CMN_ChargingMode := aC

4	PIXIT_SECC_CMN_CmValidate == cmValidate := cmValidate **and**
	PICS_CMN_CMN_ChargingMode := aC
		PIXIT_CMN_CMN_CmAmpMap := false **and**
		PICS_CMN_CMN_ChargingMode := aC

5	PIXIT_SECC_CMN_CmValidate := cmValidate **and**
	PIXIT_SECC_CMN_ArchitectureValidationNotRequired := true **and**
	PICS_CMN_CMN_ChargingMode := aC
		PICS_SECC_CMN_Pause := true **and**
		PICS_CMN_CMN_CombinedTesting := true **and**
		PIXIT_CMN_CMN_WakeUp >
		PICS_CMN_CMN_WakeUp **and**
		PICS_CMN_CMN_ChargingMode := aC

6	PIXIT_SECC_CMN_CmValidate := none_ **and**
	PICS_CMN_CMN_ChargingMode := aC
		PICS_SECC_CMN_Pause := true **and**
		PICS_CMN_CMN_CombinedTesting := true **and**
		PIXIT_CMN_CMN_WakeUp <
		PICS_CMN_CMN_WakeUp **and**
		PICS_CMN_CMN_ChargingMode := aC

7	PIXIT_CMN_CMN_CmAmpMap := false **and**
	PICS_CMN_CMN_ChargingMode := aC
		PICS_SECC_CMN_Pause := false **and**
		PICS_CMN_CMN_CombinedTesting := true **and**
		PICS_CMN_CMN_ChargingMode := aC

8	PIXIT_SECC_AC_InitialDutyCyle := dc5 **and**
	PICS_CMN_CMN_IdentificationMode := eIM **and**
	PICS_SECC_CMN_EIMDone := duringSlac **and**
	PICS_CMN_CMN_ChargingMode := aC
		PICS_SECC_CMN_Pause := true **and**
		PICS_CMN_CMN_CombinedTesting := true **and**
		PICS_CMN_CMN_WakeUp < par_SECC_T_step_X1 **and**
		PIXIT_CMN_CMN_WakeUp >
		PICS_CMN_CMN_WakeUp **and**
		PICS_CMN_CMN_ChargingMode := aC

9	PICS_SECC_CMN_Pause := true **and**
	PICS_CMN_CMN_CombinedTesting := true **and**
	PIXIT_CMN_CMN_WakeUp >
	PICS_CMN_CMN_WakeUp **and**
	PICS_CMN_CMN_ChargingMode := aC
		PICS_CMN_CMN_InitiateCmAmpMap := true **and**
		PIXIT_CMN_CMN_CmAmpMap := true **and**
		PICS_CMN_CMN_ChargingMode := aC

10	PICS_SECC_CMN_Pause := true **and**
	PICS_CMN_CMN_CombinedTesting := true **and**
	PIXIT_CMN_CMN_WakeUp <
	PICS_CMN_CMN_WakeUp **and**
	PICS_CMN_CMN_ChargingMode := aC
		PICS_CMN_CMN_InitiateCmAmpMap := false **and**
		PIXIT_CMN_CMN_CmAmpMap := true **and**
		PICS_CMN_CMN_ChargingMode := aC

11	PICS_SECC_CMN_Pause := false **and**
	PICS_CMN_CMN_CombinedTesting := true **and**
	PICS_CMN_CMN_ChargingMode := aC
		PIXIT_SECC_AC_InitialDutyCyle := dc5 **and**
		PICS_CMN_CMN_ChargingMode := aC

12	PIXIT_SECC_AC_InitialDutyCyle := dc100 **and**
	PICS_CMN_CMN_IdentificationMode := eIM **and**
		PIXIT_SECC_AC_InitialDutyCyle := dc100 **and**
		PICS_CMN_CMN_ChargingMode := aC

Configuration	EIM	PnC
	PICS_SECC_CMN_EIMDone := duringSlac **and** PICS_CMN_CMN_ChargingMode := aC	
13	PICS_SECC_CMN_Pause := true **and** PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_WakeUp < par_SECC_T_step_X1 **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC	PICS_CMN_CMN_ChargingMode := aC
14	PICS_CMN_CMN_InitiateCmAmpMap := true **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := aC	
15	PICS_CMN_CMN_InitiateCmAmpMap := false **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := aC	
16	PIXIT_SECC_AC_InitialDutyCyle := dc5 **and** PICS_CMN_CMN_IdentificationMode := eIM **and** PICS_CMN_CMN_ChargingMode := aC	
17	PIXIT_SECC_AC_InitialDutyCyle := dc5 **and** PICS_CMN_CMN_ChargingMode := aC	
18	PIXIT_SECC_AC_InitialDutyCyle := dc5 **and** PIXIT_SECC_AC_ConnectionLossHandling := optionA **and** PICS_SECC_CMN_EIMDone := afterPlugin **and** PICS_CMN_CMN_IdentificationMode := eIM **and** PICS_CMN_CMN_ChargingMode := aC	
19	PIXIT_SECC_AC_InitialDutyCyle := dc5 **and** PIXIT_SECC_AC_ConnectionLossHandling := optionB **and** PICS_SECC_CMN_EIMDone := afterPlugin **and** PICS_CMN_CMN_IdentificationMode := eIM **and** PICS_CMN_CMN_ChargingMode := aC	
20	PIXIT_SECC_AC_InitialDutyCyle := dc100 **and** PICS_CMN_CMN_ChargingMode := aC	
21	PICS_CMN_CMN_ChargingMode := aC	

Table 13 — SECC DC PICS/PIXIT configuration

Configuration	EIM	PnC
1	PIXIT_SECC_CMN_CmValidate := cmValidate **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_SECC_CMN_CmValidate := cmValidate **and** PICS_CMN_CMN_ChargingMode := dC
2	PIXIT_SECC_CMN_CmValidate := cmValidate **and** PIXIT_SECC_CMN_ArchitectureValidationNotRequired := true **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_SECC_CMN_CmValidate := cmValidate **and** PIXIT_SECC_CMN_ArchitectureValidationNotRequired := true **and** PICS_CMN_CMN_ChargingMode := dC
3	PIXIT_SECC_CMN_CmValidate := none_ **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_SECC_CMN_CmValidate := none_ **and** PICS_CMN_CMN_ChargingMode := dC
4	PIXIT_CMN_CMN_CmAmpMap := false **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_CMN_CMN_CmAmpMap := false **and** PICS_CMN_CMN_ChargingMode := dC
5	PICS_SECC_CMN_Pause := true **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC	PICS_SECC_CMN_Pause := true **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC
6	PICS_SECC_CMN_Pause := true **and** PICS_CMN_CMN_CombinedTesting := true **and**	PICS_SECC_CMN_Pause := true **and** PICS_CMN_CMN_CombinedTesting := true **and**

Configuration	EIM	PnC
	PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC
6	PICS_SECC_CMN_Pause := false **and** PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_ChargingMode := dC	PICS_SECC_CMN_Pause := false **and** PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_ChargingMode := dC
7	PICS_SECC_CMN_Pause := true **and** PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_WakeUp < par_SECC_T_step_X1 **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC	PICS_SECC_CMN_Pause := true **and** PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_WakeUp < par_SECC_T_step_X1 **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC
8	PICS_CMN_CMN_InitiateCmAmpMap := true **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC	PICS_CMN_CMN_InitiateCmAmpMap := true **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC
9	PICS_CMN_CMN_InitiateCmAmpMap := false **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC	PICS_CMN_CMN_InitiateCmAmpMap := false **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC
10	PICS_CMN_CMN_ChargingMode := dC	PICS_CMN_CMN_ChargingMode := dC

Table 14 — EVCC AC PICS/PIXIT configuration

Configuration	EIM	PnC
1	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PICS_CMN_CMN_ChargingMode := aC
2	PIXIT_EVCC_CMN_CmValidate := none_ **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := none_ **and** PICS_CMN_CMN_ChargingMode := aC
3	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := skip **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := skip **and** PICS_CMN_CMN_ChargingMode := aC
4	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequire d := continue_ **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequire d := continue_ **and** PICS_CMN_CMN_ChargingMode := aC
5	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequire d := skip **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequire d := skip **and** PICS_CMN_CMN_ChargingMode := aC
6	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := retry **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := retry **and** PICS_CMN_CMN_ChargingMode := aC
7	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC
8	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC
9	PIXIT_EVCC_CMN_Pause := none_ **and**	PIXIT_EVCC_CMN_Pause := none_ **and**

Configuration	EIM	PnC
	PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_ChargingMode := aC	PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_ChargingMode := aC
10	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepWithoutCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepWithoutCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC
11	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepAfterCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepAfterCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := aC
12	PIXIT_CMN_CMN_CmAmpMap := false **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_CMN_CMN_CmAmpMap := false **and** PICS_CMN_CMN_ChargingMode := aC
13	PICS_CMN_CMN_InitiateCmAmpMap := true **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := aC	PICS_CMN_CMN_InitiateCmAmpMap := true **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := aC
14	PICS_CMN_CMN_InitiateCmAmpMap := false **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := aC	PICS_CMN_CMN_InitiateCmAmpMap := false **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := aC
15	PIXIT_EVCC_CMN_TTMatchingRepetitionConfig := true **and** PIXIT_EVCC_CMN_TTMatchingRepetition **and** PIXIT_EVCC_CMN_TTMatchingRate **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_TTMatchingRepetitionConfig := true **and** PIXIT_EVCC_CMN_TTMatchingRepetition **and** PIXIT_EVCC_CMN_TTMatchingRate **and** PICS_CMN_CMN_ChargingMode := aC
16	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := continue_ **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := continue_ **and** PICS_CMN_CMN_ChargingMode := aC
17	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := terminate **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := terminate **and** PICS_CMN_CMN_ChargingMode := aC
18	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := iterate **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := iterate **and** PICS_CMN_CMN_ChargingMode := aC
19	PIXIT_EVCC_AC_ConnectionLossHandling == optionA **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_AC_ConnectionLossHandling == optionA **and** PICS_CMN_CMN_ChargingMode := aC
20	PIXIT_EVCC_AC_ConnectionLossHandling == optionB **and** PIXIT_EVCC_AC_TconnResetup **and** PICS_CMN_CMN_ChargingMode := aC	PIXIT_EVCC_AC_ConnectionLossHandling == optionB **and** PIXIT_EVCC_AC_TconnResetup **and** PICS_CMN_CMN_ChargingMode := aC
21	PICS_CMN_CMN_IdentificationMode := eIM **and** PICS_CMN_CMN_ChargingMode := aC	PICS_CMN_CMN_ChargingMode := aC
22	PICS_CMN_CMN_ChargingMode := aC	

Table 15 — EVCC DC PICS/PIXIT configuration

Configuration	EIM	PnC
1	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PICS_CMN_CMN_ChargingMode := dC
2	PIXIT_EVCC_CMN_CmValidate := none_ **and**	PIXIT_EVCC_CMN_CmValidate := none_ **and**

Configuration	EIM	PnC
	PICS_CMN_CMN_ChargingMode := dC	PICS_CMN_CMN_ChargingMode := dC
3	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := skip **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := skip **and** PICS_CMN_CMN_ChargingMode := dC
4	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequired := continue_ **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequired := continue_ **and** PICS_CMN_CMN_ChargingMode := dC
5	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequired := skip **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationNotRequired := skip **and** PICS_CMN_CMN_ChargingMode := dC
6	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := retry **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := retry **and** PICS_CMN_CMN_ChargingMode := dC
7	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC
8	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_Pause := pause **and** PICS_CMN_CMN_CombinedTesting := true **and** PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC
9	PIXIT_EVCC_CMN_Pause := none_ **and** PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_Pause := none_ **and** PICS_CMN_CMN_CombinedTesting := true **and** PICS_CMN_CMN_ChargingMode := dC
10	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepWithoutCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepWithoutCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC
11	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepAfterCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC	PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepAfterCharge **and** PICS_CMN_CMN_CombinedTesting := true **and** par_V2G_SECC_Pmax0W < PICS_CMN_CMN_WakeUp **and** PICS_CMN_CMN_ChargingMode := dC
12	PIXIT_CMN_CMN_CmAmpMap := false **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_CMN_CMN_CmAmpMap := false **and** PICS_CMN_CMN_ChargingMode := dC
13	PICS_CMN_CMN_InitiateCmAmpMap := true **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC	PICS_CMN_CMN_InitiateCmAmpMap := true **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC
14	PICS_CMN_CMN_InitiateCmAmpMap := false **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC	PICS_CMN_CMN_InitiateCmAmpMap := false **and** PIXIT_CMN_CMN_CmAmpMap := true **and** PICS_CMN_CMN_ChargingMode := dC
15	PIXIT_EVCC_CMN_TTMatchingRepetitionConfig := true **and** PIXIT_EVCC_CMN_TTMatchingRepetition **and** PIXIT_EVCC_CMN_TTMatchingRate **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_TTMatchingRepetitionConfig := true **and** PIXIT_EVCC_CMN_TTMatchingRepetition **and** PIXIT_EVCC_CMN_TTMatchingRate **and** PICS_CMN_CMN_ChargingMode := dC
16	PIXIT_EVCC_CMN_CmValidate := cmValidate **and**	PIXIT_EVCC_CMN_CmValidate := cmValidate **and**

Configuration	EIM	PnC
	PIXIT_EVCC_CMN_FallbackValidationFailed := continue_ **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_FallbackValidationFailed := continue_ **and** PICS_CMN_CMN_ChargingMode := dC
17	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := terminate **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_FallbackValidationFailed := terminate **and** PICS_CMN_CMN_ChargingMode := dC
18	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := iterate **and** PICS_CMN_CMN_ChargingMode := dC	PIXIT_EVCC_CMN_CmValidate := cmValidate **and** PIXIT_EVCC_CMN_ConcurrentValidation := iterate **and** PICS_CMN_CMN_ChargingMode := dC
19	PICS_CMN_CMN_ChargingMode := dC	PICS_CMN_CMN_ChargingMode := dC

7.4 Test suite identifiers

The selection of common naming conventions is one simple and often used mechanism to implement test suites which are consistent, maintainable and understandable for multiple users. Therefore, based on common ETSI naming conventions more specific naming conventions for the ISO 15118 conformance test suite are defined.

7.4.1 Module identifiers

All modules defined in the TSS start with a capital letter. The ISO 15118 test suite specific module identifier for template, function or test case modules is defined as:

<modtype>_<sut>_<ctx>

The segments of this identifier are defined in Table 16. An example for test case module identifier is:

TestCases_SECC_CmSlacParm

Table 16 — ISO 15118 test suite naming convention for modules

Identifier	Values	Description
<modtype>		Module type
	TestCases	Module including test cases
	Functions	Module including functions
	Templates	Module including templates
<sut>		System under test
	EVCC	Electric Vehicle Communication Controller
	SECC	Supply Equipment Communication Controller
	CMN	Common (exclusively for template modules)
<ctx>	{fullname}	Context (e.g. name of message pattern signal name according to standard)

NOTE 1 For module types other than templates, functions or test cases there is no identifier format defined.

7.4.2 Test case identifiers

The naming conventions for test cases are using a prefix, which is defined by ETSI as shown in Table 17.

Table 17 — ETSI naming convention for test case names

Keyword	Definition	Example
testcase	Every testcase begins with TC (TC_)	TC_CmSlacParm

name	TC_TestCaseName	

The ISO 15118 test suite specific test case identifier is defined as:

TC_<sut>_<dom>_<ttyp>_<ctx>_<nn>

The segments of this identifier are described in Table 18. An example for test case identifier is:

TC_SECC_CMN_VTB_CmSlacParm_001

Table 18 — ISO 15118 test suite naming convention for test case identifiers

Identifier	Values	Description
<prefix>	TC	see Table 17
<sut>		System under test
	EVCC	Electric Vehicle Communication Controller
	SECC	Supply Equipment Communication Controller
<dom>		Domain
	AC	AC specific behaviors
	DC	DC specific behaviors
	CMN	Common
<ttyp>		Type of testing
	VTB	Valid test behavior
	ITB	Invalid test behavior
<ctx>	{fullname}	Context (e.g. name of message pattern signal name according to standard)
<nn>	{xxx}	Sequential number from 001 to 999

7.4.3 Template identifiers

The naming conventions for templates are using a prefix, which is defined by ETSI as shown in Table 19.

Table 19 — ETSI naming convention for templates

Keyword	Context	Definition	Example
template name	Templates with concrete attribute values	Every template begins with keyword m (m_)	m_DNSRequest
template name	Templates with wildcards or matching expression	If a template contains or refers to templates with wildcards {*|?} then template name begins with keyword mw (mw_)	mw_DNSResponse
template name	Templates with parameters, which do not assign or refer to templates with wildcards or matching expression	If a template contains attributes which are defined by parameters or constant values, then template name begins with keyword md (md_)	md_DNSResponse (integer ip)
template name	Templates with parameters, which do assign or refer to templates with wildcards or matching expression	If a template contains attributes which are defined by parameters, constant values or wildcards, then template name begins with keyword mdw (mdw_)	mdw_DNSResponse (integer ip)

The ISO 15118 test suite specific template identifier is defined as:

<prefix>_<sut>_<dom>_<dtyp>_<nn>

The segments of this identifier are described in Table 20. An example for template identifier is:

md_EVCC_CMN_CmSlacParm_001

Table 20 — ISO 15118 test suite naming convention for template identifiers

Identifier	Values	Description
<prefix>		Type of template (see Table 19)
<sut>		System under test
	EVCC	Electric Vehicle Communication Controller
	SECC	Supply Equipment Communication Controller
	CMN	Common
<dom>		Domain
	AC	AC specific behaviors
	DC	DC specific behaviors
	CMN	Common
<dtyp>	{fullname}	Label of (root) data type according to standard
<nn>	{xxx}	Sequential number from 001 to 999

7.4.4 Function identifiers

The naming conventions for functions are using a prefix, which is defined by ETSI as shown in Table 21.

Table 21 — ETSI naming convention for function names

Keyword	Context	Definition	Example
function name	All functions	Every function begins with f (f_)	f_functionName

The ISO 15118 test suite naming convention for test case functions is defined as:

<prefix>_<sut>_<dom>_<ttyp>_<ctx>_<nn>

The segments of this identifier are described in Table 22. An example for template identifier is:

f_EVCC_CMN_ VTB_CmSlacParm_001

Table 22 — ISO 15118 test suite naming convention for function names

Identifier	Values	Description
<prefix>		f_ (see Table 21)
<sut>		System under test
	EVCC	Electric Vehicle Communication Controller
	SECC	Supply Equipment Communication Controller
<dom>		Domain
	AC	AC specific behaviors
	DC	DC specific behaviors
	CMN	Common
<ttyp>		Type of testing

Identifier	Values	Description
	VTB	Valid test behavior
	ITB	Invalid test behavior
<ctx>	{fullname}	Context (e.g. name of message pattern signal name according to standard)
<nn>	{xxx}	Sequential number from 001 to 999

7.4.5 Timer identifiers

The naming conventions for timers are using a prefix, which is defined by ETSI as shown in Table 23.

Table 23 — ETSI naming convention for timers

Keyword	Context	Definition	Example
timer name	t_	Local timer	t_wait
timer name	tc_	Timer defined within a component	tc_authMin

The ISO 15118 test suite specific timer identifier is defined as:

<prefix>_<ctx>

The segments of this identifier are described in Table 24. An example for timer identifier is:

tc_V2G_EVCC_Msg_Timer

Table 24 — ISO 15118 test suite naming convention for timer identifiers

Identifier	Values	Description
<prefix>		Type of timer (see Table 23)
<ctx>	{fullname}	Context (e.g. name of timer according to ISO 15118-3 or if not part of the standard any given name describing the context of the timer)

7.4.6 PICS/PIXIT identifiers

The ISO 15118 test suite naming convention for PICS/PIXIT is defined as:

<pic>_<sut>_<dom>_<ctx>

The segments of this identifier are described in Table 25. An example for PICS/PIXIT identifier is:

PICS_SECC_CMN_CmValidate

Table 25 — ISO 15118 test suite naming convention for PICS/PIXIT identifiers

Identifier	Values	Description
<pic>		Protocol implementation capability
	PICS	Protocol Implementation Conformance Statement
	PIXIT	Protocol Implementation extra Information for Testing
<sut>		System under test
	EVCC	Electric Vehicle Communication Controller
	SECC	Supply Equipment Communication Controller
	CMN	Common
<dom>		Domain

	AC	AC specific behaviors
	DC	DC specific behaviors
	CMN	Common
<ctx>	{fullname}	Context (e.g. name of message pattern signal name according to standard)

7.4.7 Verdict identifiers

In this sub clause the conventions for test verdicts are defined. The test verdicts defined in this document are listed in Table 26.

Table 26 — ISO 15118 test suite conventions on verdict handling

Verdict type	TTCN-3 Definition	ISO 15118 Test Suite
none	Is implicitly assigned in the beginning of every test case by default and is reported as a final verdict in the absence of any other verdict assignment during the test case execution.	No TSS specific definition (see TTCN-3 definition).
pass	Means that everything is OK. A verdict given when the observed outcome satisfies the test purpose and is valid with respect to the relevant requirements and with respect to the PICS. [ITU-T X.290]	If in review of a requirement the SUT has a correct behavior, then this verdict type shall be used.
inconc	A verdict given when the observed outcome is valid with respect to the relevant requirements but prevents the test purpose from being accomplished. [ITU-T X.290]	Means that neither pass nor fail can be reliably assigned.
fail	A verdict given when the observed outcome is syntactically invalid or inopportune with respect to the relevant requirements or the PICS/PIXIT. [ITU-T X.290]	If in review of a requirement the SUT has a wrong behavior, then this verdict type shall be used.

7.5 Test suite coverage

The following conditions apply in terms of test case coverage for this document regarding requirements defined in ISO 15118-3:

— Requirements that are out of scope according to the conventions defined in this document or cannot be tested in black-box tests are summarized in Table 27.

— Requirements that are not explicitly tested through a dedicated test case in the ATS but implicitly tested by the SUT Adapter/Codec are summarized in Table 28.

Table 27 — Requirements of ISO 15118-3 not considered in the ATS

List of requirement IDs	Comment (no consideration in ATS)
[V2G3-M09-05], [V2G3-M09-09], [V2G3-A06-02], [V2G3-A09-06], [V2G3-A09-20], [V2G3-A09-21], [V2G3-A09-22], [V2G3-A09-26], [V2G3-A09-27], [V2G3-A09-29], [V2G3-A09-48], [V2G3-A09-49], [V2G3-A09-69], [V2G3-A09-100], [V2G3-A09-102]	Cannot be tested explicitly in EVCC black box test configuration
[V2G3-M06-10], [V2G3-A06-03], [V2G3-A09-19], [V2G3-A09-88], [V2G3-A09-92], [V2G3-A09-103],	Cannot be tested explicitly in SECC black box test configuration

List of requirement IDs	Comment (no consideration in ATS)
[V2G3-A09-104]	
[V2G3-M08-02], [V2G3-M09-18], [V2G3-A06-04], [V2G3-A09-119], [V2G3-A09-120]	Cannot be tested explicitly in SECC/EVCC black box test configuration
[V2G3-A11-08], [V2G3-A11-09], [V2G3-A11-10], [V2G3-A11-11], [V2G3-A11-12], [V2G3-A11-13], [V2G3-A11-14], [V2G3-A11-15]	Out of scope of the ATS (The requirements for calibrating the power level of the PLC signal are not considered in the ATS -> see sub clause 8.1)
[V2G3-M06-02], [V2G3-M06-03], [V2G3-M06-12], [V2G3-M06-14], [V2G3-A09-117]	Out of scope of this standard (Requirement refers to ISO 15118-2. Cannot be tested in the context of ISO 15118-3)
[V2G3-M06-01], [V2G3-A09-02], [V2G3-A09-93]	Out of scope of this standard (Requirement refers to external RFCs, standards, etc.)
[V2G3-M06-16], [V2G3-A06-01], [V2G3-A09-107], [V2G3-A10-01], [V2G3-A11-01], [V2G3-A11-02], [V2G3-A11-03], [V2G3-A11-04], [V2G3-A11-05], [V2G3-A11-06], [V2G3-A11-07]	Out of scope of this standard (Requirements may be validated with dedicated measurement equipment but are not considered in the ATS)
[V2G3-B09-01], [V2G3-B09-02], [V2G3-B09-03], [V2G3-B09-04], [V2G3-B11-01], [V2G3-B11-02], [V2G3-B11-03], [V2G3-B11-04], [V2G3-B11-05]	Out of scope of this standard (Requirements on IEEE 1901.2 G3-PLC are not covered in this standard)

Table 28 — Requirements of ISO 15118-3 implicitly covered by the SUT adapters / codecs

List of requirement IDs	Comment (implicitly tested)
[V2G3-A09-108]	Tested implicitly by ISO 15118-3 SUT adapter (refer to Part 5 reqs.)
[V2G3-A09-24], [V2G3-A09-55]	Tested implictly by ISO 15118-3 SUT adapter

The resulting test suite coverage in this document with reference to requirements in ISO 15118-3 is summarized in Table 29. It only includes requirements which are not listed for exclusion in Table 27 or Table 28. Table 29 defines the relevance of requirements of ISO 15118-3 for the ATS, based upon the type of SUT (EVCC or SECC), the authentication profile (EIM or PnC) and AC or DC charging. The following symbols are used in Table 29:

X	Indicates requirements that are covered in the ATS with one or more test cases.
I	Indicates requirements that are indirectly covered in the ATS but other than those listed in Table 29 can be directly associated to one or more test cases in the ATS.
P	Indicates requirements that are only partially covered by one or more test cases in the ATS.
N	Requirements that are not testable for the profile defined in the respective column.
-	Requirements that are not applicable for the profile defined in the respective column.
O	Requirements that are out of scope for the profile defined in the respective column.

Table 29 — ATS coverage of requirements in ISO 15118-3

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
[V2G3-M06-04]	X	O	O	O	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered for the profile defined in the respective column. Out of scope for the profile defined in the respective column (Requirement

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									refers to ISO 15118-2. Cannot be tested in the context of ISO 15118-3) TC_SECC_AC_VTB_CmSlacParm_001 TC_SECC_AC_VTB_CmSlacParm_002 TC_SECC_AC_VTB_CmSlacParm_003 TC_SECC_AC_VTB_CmSlacParm_004
[V2G3-M06-05]	X	—	—	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_AC_VTB_CmSlacParm_001
[V2G3-M06-06]	X	—	—	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_AC_VTB_PLCLinkStatus_001
[V2G3-M06-07]	X	—	—	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_AC_VTB_CmSlacParm_002
[V2G3-M06-08]	X	—	—	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_AC_VTB_CmSlacParm_001 TC_SECC_AC_VTB_CmSlacParm_003
[V2G3-M06-09]	I	—	—	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-M06-05] or [V2G3-M06-07] TC_SECC_AC_VTB_CmSlacParm_001 TC_SECC_AC_VTB_CmSlacParm_002 TC_SECC_AC_VTB_CmSlacParm_003
[V2G3-M06-11]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-15] Group_[V2G3-M06-11]
[V2G3-M06-13]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-05] Group_[V2G3-A09-05]_[V2G3-M06-13]
[V2G3-M06-15]	—	—	—	—	P	—	—	—	Requirement definition considers various possible scenarios, so the requirement can only be tested partially for exemplarily selected cases TC_EVCC_AC_VTB_AttenuationCharacterization_001 TC_EVCC_AC_VTB_AttenuationCharacterization_002
[V2G3-M07-01]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-15] Group_[V2G3-M07-01]
[V2G3-M07-02]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-15] TC_SECC_CMN_VTB_CmSlacParm_009
[V2G3-M07-03]	I	I	I	I	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-M07-05] and [V2G3-M07-15]

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									Group_[V2G3-M07-03]
[V2G3-M07-04]	P	—	P	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Internal processing cannot be tested in SECC black box test configuration, so the requirement can only be tested partially TC_SECC_AC_VTB_PLCLinkStatus_011
[V2G3-M07-05]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Internal processing cannot be tested in SECC black box test configuration, so the requirement can only be tested partially TC_SECC_AC_VTB_PLCLinkStatus_005 TC_SECC_AC_VTB_PLCLinkStatus_006 TC_SECC_DC_VTB_PLCLinkStatus_004
[V2G3-M07-06]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered (AC). Internal states cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially TC_SECC_AC_VTB_PLCLinkStatus_005 TC_SECC_AC_VTB_PLCLinkStatus_006 TC_SECC_DC_VTB_PLCLinkStatus_004
[V2G3-M07-07]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered (AC). Internal states cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially TC_SECC_AC_VTB_PLCLinkStatus_005 TC_SECC_AC_VTB_PLCLinkStatus_006 TC_SECC_DC_VTB_PLCLinkStatus_004
[V2G3-M07-08]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered Group_[V2G3-M07-08]
[V2G3-M07-09]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Internal processing cannot be tested in SECC black box test configuration, so the requirement can only be tested partially TC_SECC_AC_VTB_PLCLinkStatus_005 TC_SECC_AC_VTB_PLCLinkStatus_006 TC_SECC_AC_VTB_PLCLinkStatus_011 TC_SECC_DC_VTB_PLCLinkStatus_004
[V2G3-M07-10]	I	—	—	—	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-M07-11] and [V2G3-M07-12] TC_SECC_AC_VTB_PLCLinkStatus_006 TC_SECC_AC_VTB_PLCLinkStatus_007
[V2G3-M07-11]	X	—	—	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered (see RQs [V2G3-M07-05] - [V2G3-M07-10]) TC_SECC_AC_VTB_PLCLinkStatus_006
[V2G3-M07-12]	P	—	—	—	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Internal states and processing cannot be tested explicitly in SECC black box configuration, so the requirement can only be

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									tested partially TC_SECC_AC_VTB_PLCLinkStatus_007
[V2G3-M07-13]	—	—	—	—	P	P	P	P	Internal states cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_PLCLinkStatus_006 TC_EVCC_CMN_VTB_PLCLinkStatus_007
[V2G3-M07-14]	—	—	—	—	I	—	I	—	Tested indirectly by consideration of requirement [V2G3-M07-15] and [V2G3-M07-16] TC_EVCC_AC_VTB_PLCLinkStatus_003 TC_EVCC_AC_VTB_PLCLinkStatus_004 TC_EVCC_AC_VTB_PLCLinkStatus_005 TC_EVCC_AC_VTB_PLCLinkStatus_007 TC_EVCC_AC_VTB_PLCLinkStatus_008
[V2G3-M07-15]	—	—	—	—	X	—	X	—	Requirement is only testable if a corresponding PIXIT is considered (see RQ [V2G3-M07-13]) TC_EVCC_AC_VTB_PLCLinkStatus_003 TC_EVCC_AC_VTB_PLCLinkStatus_007
[V2G3-M07-16]	—	—	—	—	P	—	P	—	Requirement is only testable if a corresponding PIXIT is considered. Internal states and processing cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_AC_VTB_PLCLinkStatus_004 TC_EVCC_AC_VTB_PLCLinkStatus_005 TC_EVCC_AC_VTB_PLCLinkStatus_008
[V2G3-M07-17]	—	—	—	—	X	—	X	—	Requirement is only testable if a corresponding PIXIT is considered (Only min timer defined) TC_EVCC_AC_VTB_PLCLinkStatus_004 TC_EVCC_AC_VTB_PLCLinkStatus_005 TC_EVCC_AC_VTB_PLCLinkStatus_008
[V2G3-M07-18]	—	—	—	—	X	—	X	—	Requirement is only testable if a corresponding PIXIT is considered TC_EVCC_AC_VTB_PLCLinkStatus_005 TC_EVCC_AC_VTB_PLCLinkStatus_008
[V2G3-M07-19]	—	—	—	—	P	P	P	P	Stored logical network parameter are indirectly testable under consideration of [V2G3-M07-21]. Internal processing cannot be tested explicitly in EVCC black box test configuration, so the requirement can only be tested partially. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. Group_[V2G3-M07-19]_[V2G3-M07-29]
[V2G3-M07-20]	P	P	P	P	—	—	—	—	Stored logical network parameter are indirectly testable under consideration of [V2G3-M07-21]. Internal processing cannot be tested explicitly in EVCC black box test configuration, so the requirement can only be tested partially. Only

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. Group_[V2G3-M07-20]
[V2G3-M07-21]	I	I	I	I	I	I	I	I	For EV side only indirectly testable if the SDP process is executed. For EVSE side indirectly testable under consideration [V2G3-M07-24]. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. Group_[V2G3-M07-21]
[V2G3-M07-22]	I	I	I	I	I	I	I	I	Indirectly testable under consideration [V2G3-M07-21]. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. TC_EVCC_AC_VTB_PLCLinkStatus_001 TC_EVCC_DC_VTB_PLCLinkStatus_001 TC_SECC_AC_VTB_PLCLinkStatus_002 TC_SECC_DC_VTB_PLCLinkStatus_001
[V2G3-M07-23]	I	I	I	I	I	I	I	I	Indirectly testable under consideration [V2G3-M07-21]. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. TC_EVCC_AC_VTB_PLCLinkStatus_001 TC_EVCC_DC_VTB_PLCLinkStatus_001 TC_SECC_AC_VTB_PLCLinkStatus_002 TC_SECC_DC_VTB_PLCLinkStatus_001
[V2G3-M07-24]	X	X	X	X	—	—	—	—	Last known parameter set can only be tested indirectly under consideration of [V2G3-M07-21]. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. TC_SECC_AC_VTB_PLCLinkStatus_003 TC_SECC_DC_VTB_PLCLinkStatus_002
[V2G3-M07-25]	X	X	X	X	—	—	—	—	Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. TC_SECC_AC_VTB_PLCLinkStatus_010 TC_SECC_AC_VTB_PLCLinkStatus_012 TC_SECC_DC_VTB_PLCLinkStatus_006 TC_SECC_DC_VTB_PLCLinkStatus_007
[V2G3-M07-26]	I	I	I	I	—	—	—	—	Last known parameter set can only be tested indirectly under consideration of [V2G3-M07-21]. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. TC_SECC_AC_VTB_PLCLinkStatus_002 TC_SECC_DC_VTB_PLCLinkStatus_001
[V2G3-M07-27]	X	X	X	X	—	—	—	—	Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT.

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									TC_SECC_AC_VTB_PLCLinkStatus_004 TC_SECC_DC_VTB_PLCLinkStatus_003
[V2G3-M07-28]	—	—	—	—	X	X	X	X	Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. Group_[V2G3-M07-28]
[V2G3-M07-29]	—	—	—	—	I	I	I	I	Indirectly testable under consideration [V2G3-M07-21]. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. Group_[V2G3-M07-19]_[V2G3-M07-29]
[V2G3-M07-30]	—	—	—	—	I	I	I	I	Indirectly testable under consideration [V2G3-M07-21]. Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. TC_EVCC_AC_VTB_PLCLinkStatus_001 TC_EVCC_DC_VTB_PLCLinkStatus_001
[V2G3-M07-31]	X	X	X	X	—	—	—	—	Only testable under consideration of ISO 15118-4 TCs as Pre-Condition with the consideration of a the corresponding PIXIT. Group_[V2G3-M07-31]_[V2G3-M07-32]
[V2G3-M07-32]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Group_[V2G3-M07-31]_[V2G3-M07-32]
[V2G3-M07-33]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. TC_SECC_AC_VTB_PLCLinkStatus_003 TC_SECC_AC_VTB_PLCLinkStatus_012 TC_SECC_DC_VTB_PLCLinkStatus_002 TC_SECC_DC_VTB_PLCLinkStatus_007
[V2G3-M07-34]	P	P	P	P	P	P	P	P	Internal states cannot be tested explicitly in SECC/EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_PLCLinkStatus_008 TC_SECC_CMN_VTB_PLCLinkStatus_005 TC_SECC_AC_VTB_PLCLinkStatus_006 TC_SECC_AC_VTB_PLCLinkStatus_007
[V2G3-M08-01]	P	P	P	P	P	P	P	P	Table 3 can only be tested partially TC_EVCC_CMN_VTB_CmValidate_001 TC_SECC_CMN_VTB_PLCLinkStatus_004
[V2G3-M09-01]	X	X	X	X	—	—	—	—	TC_SECC_CMN_VTB_PLCLinkStatus_004
[V2G3-M09-02]	I	I	I	I	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-45] or [V2G3-A09-37] TC_EVCC_CMN_VTB_AttenuationCharacterization_001 TC_EVCC_CMN_VTB_AttenuationCharacterization_002 TC_SECC_CMN_VTB_AttenuationCharacterization_

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									001 TC_SECC_CMN_VTB_AttenuationCharacterization_003 TC_SECC_CMN_VTB_AttenuationCharacterization_020
[V2G3-M09-03]	N	N	N	N	X	X	X	X	Cannot be tested explicitly in SECC black box test configuration for the profile defined in the respective column TC_EVCC_CMN_VTB_AttenuationCharacterization_001 TC_EVCC_CMN_VTB_AttenuationCharacterization_002
[V2G3-M09-04]	X	X	X	X	N	N	N	N	Cannot be tested explicitly in EVCC black box test configuration for the profile defined in the respective column TC_SECC_CMN_VTB_AttenuationCharacterization_001 TC_SECC_CMN_VTB_AttenuationCharacterization_003 TC_SECC_CMN_VTB_AttenuationCharacterization_020
[V2G3-M09-06]	—	—	—	—	X	X	X	X	TC_EVCC_CMN_VTB_CmValidate_001 TC_EVCC_CMN_VTB_CmValidate_002 TC_EVCC_CMN_VTB_CmValidate_003
[V2G3-M09-07]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Only the start of the validation can be tested, so the Requirement can only be tested partially Group_[V2G3-M09-07]_[V2G3-A09-57]
[V2G3-M09-08]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmValidate_013 TC_EVCC_CMN_VTB_CmValidate_015
[V2G3-M09-10]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-87] TC_SECC_CMN_VTB_CmValidate_001 TC_SECC_CMN_VTB_CmValidate_009 TC_SECC_CMN_VTB_CmValidate_011 TC_SECC_CMN_VTB_CmValidate_012
[V2G3-M09-11]	—	—	—	—	X	X	X	X	TC_EVCC_CMN_VTB_CmValidate_001 TC_EVCC_CMN_VTB_CmValidate_002 TC_EVCC_CMN_VTB_CmValidate_003
[V2G3-M09-12]	P	P	P	P	P	P	P	P	Only a part of the message sequence will be considered, so the requirement can only be tested partially Group_[V2G3-M09-12]_[V2G3-A09-54]
[V2G3-M09-13]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									PIXIT is considered TC_SECC_CMN_VTB_CmValidate_009
[V2G3-M09-14]	—	—	—	—	X	X	X	X	Requirement is only testable if a corresponding PIXIT is considered TC_EVCC_CMN_VTB_CmValidate_001 TC_EVCC_CMN_VTB_CmValidate_002 TC_EVCC_CMN_VTB_CmValidate_003
[V2G3-M09-15]	N	N	N	N	P	P	P	P	Cannot be tested explicitly in SECC black box test configuration for the profile defined in the respective column due to strong configuration dependency on the SUT. Can only be tested partially with exemplarily parameterized duty cycles for the profile defined in the respective column TC_EVCC_CMN_VTB_CmValidate_002 TC_EVCC_CMN_VTB_CmValidate_003
[V2G3-M09-16]	I	I	I	I	I	I	I	I	Tested indirectly by consideration of the SDP process within sleep mode configuration. Requirement is only testable if a corresponding PIXIT is considered Group_[V2G3-M09-16]
[V2G3-M09-17]	P	P	P	P	P	P	P	P	Internal states and processing cannot be tested explicitly in SECC/EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_AC_VTB_PLCLinkStatus_006 TC_EVCC_DC_VTB_PLCLinkStatus_003 TC_SECC_AC_VTB_PLCLinkStatus_008 TC_SECC_DC_VTB_PLCLinkStatus_005
[V2G3-M09-19]	P	P	P	P	P	P	P	P	Internal states and processing cannot be tested explicitly in SECC/EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_PLCLinkStatus_002 TC_SECC_CMN_VTB_PLCLinkStatus_002
[V2G3-M12-01]	I	I	I	I	I	I	I	I	Tested indirectly by consideration of the SDP process within sleep mode configuration. Requirement is only testable if a corresponding PIXIT is considered Group_[V2G3-M12-01]
[V2G3-A06-05]	—	—	—	—	P	P	P	P	Requirement definition considers various possible scenarios, so the requirement can only be tested partially for exemplarily selected cases TC_EVCC_CMN_VTB_CmSlacParm_009
[V2G3-A08-01]	P	P	P	P	P	P	P	P	Table A.1 can only be tested partially Group_[V2G3-A08-01]
[V2G3-A09-01]	P	P	P	P	P	P	P	P	Only a part of the message sequence will be considered, so the requirement can only be tested partially Group_[V2G3-A09-01]_[V2G3-A09-17]

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
[V2G3-A09-03]	P	P	P	P	—	—	—	—	Internal states cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-03]
[V2G3-A09-04]	P	P	P	P	P	P	P	P	Table A.2 can only be tested partially Group_[V2G3-A09-04]

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
[V2G3-A09-05]	—	—	—	—	P	P	P	P	Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-05]_[V2G3-M06-13]
[V2G3-A09-07]	—	—	—	—	P	P	P	P	Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-07]_[V2G3-A09-08]_[V2G3-A09-10]
[V2G3-A09-08]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-10] Group_[V2G3-A09-07]_[V2G3-A09-08]_[V2G3-A09-10]
[V2G3-A09-09]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-10] Group_[V2G3-A09-09]
[V2G3-A09-10]	—	—	—	—	P	P	P	P	Internal states and timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-07]_[V2G3-A09-08]_[V2G3-A09-10]
[V2G3-A09-11]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-15] Group_[V2G3-A09-11]_[V2G3-A09-15]
[V2G3-A09-12]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-13] TC_SECC_CMN_VTB_CmSlacParm_004 TC_SECC_CMN_VTB_CmSlacParm_005 TC_SECC_CMN_VTB_CmSlacParm_006
[V2G3-A09-13]	P	P	P	P	—	—	—	—	Internal states and timings cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially TC_SECC_CMN_VTB_CmSlacParm_004 TC_SECC_CMN_VTB_CmSlacParm_005 TC_SECC_CMN_VTB_CmSlacParm_006
[V2G3-A09-14]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-13] TC_SECC_CMN_VTB_CmSlacParm_005 TC_SECC_CMN_VTB_CmSlacParm_006
[V2G3-A09-15]	P	P	P	P	—	—	—	—	Internal timings cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-11]_[V2G3-A09-15]
[V2G3-A09-16]	P	P	P	P	—	—	—	—	Requirement definition considers various possible scenarios, so the requirement can only be tested partially for exemplarily selected cases TC_SECC_CMN_VTB_AttenuationCharacterization_002

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
[V2G3-A09-17]	P	P	P	P	P	P	P	P	Only a part of the message sequence will be considered, so the requirement can only be tested partially Group_[V2G3-A09-01]_[V2G3-A09-17]
[V2G3-A09-18]	P	P	P	P	P	P	P	P	Only a part of the message sequence will be considered, so the requirement can only be tested partially Group_[V2G3-A09-18]
[V2G3-A09-23]	P	P	P	P	P	P	P	P	Table A.4 can only be tested partially Group_[V2G3-A09-23]
[V2G3-A09-25]	—	—	—	—	P	P	P	P	Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-25]_[V2G3-A09-28]
[V2G3-A09-28]	—	—	—	—	P	P	P	P	Internal states cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-25]_[V2G3-A09-28]
[V2G3-A09-30]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-123] Group_[V2G3-A09-30]_[V2G3-A09-31]_[V2G3-A09-32]
[V2G3-A09-31]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-123] Group_[V2G3-A09-30]_[V2G3-A09-31]_[V2G3-A09-32]
[V2G3-A09-32]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-123] Group_[V2G3-A09-30]_[V2G3-A09-31]_[V2G3-A09-32]
[V2G3-A09-33]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-37] TC_EVCC_CMN_VTB_AttenuationCharacterization_002
[V2G3-A09-34]	—	—	—	—	P	P	P	P	TC_EVCC_CMN_VTB_CmSlacMatch_012
[V2G3-A09-35]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-123] Group_[V2G3-A09-35]
[V2G3-A09-36]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-123] TC_EVCC_CMN_VTB_AttenuationCharacterization_011
[V2G3-A09-37]	—	—	—	—	P	P	P	P	Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_AttenuationCharacterization_001 TC_EVCC_CMN_VTB_AttenuationCharacterization_002

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									TC_EVCC_AC_VTB_AttenuationCharacterization_001 TC_EVCC_AC_VTB_AttenuationCharacterization_002
[V2G3-A09-38]	—	—	—	—	P	P	P	P	Only one case can be considered, so the requirement can only be tested partially Group_[V2G3-A09-38]
[V2G3-A09-39]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-40] TC_SECC_CMN_VTB_AttenuationCharacterization_012
[V2G3-A09-40]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-40] TC_SECC_CMN_VTB_AttenuationCharacterization_012
[V2G3-A09-41]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-41] Group_[V2G3-A09-41]
[V2G3-A09-42]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-45] Group_[V2G3-A09-42]_[V2G3-A09-43]
[V2G3-A09-43]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-45] Group_[V2G3-A09-42]_[V2G3-A09-43]
[V2G3-A09-44]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-45] Group_[V2G3-A09-44]_[V2G3-A09-45]
[V2G3-A09-45]	P	P	P	P	—	—	—	—	Internal timings cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-44]_[V2G3-A09-45]
[V2G3-A09-46]	P	P	P	P	—	—	—	—	Internal states cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-46]
[V2G3-A09-47]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-46] Group_[V2G3-A09-47]
[V2G3-A09-50]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmValidate_016 TC_EVCC_CMN_VTB_CmValidate_017
[V2G3-A09-51]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									TC_EVCC_CMN_VTB_CmValidate_013 TC_EVCC_CMN_VTB_CmValidate_014 TC_EVCC_CMN_VTB_CmValidate_015
[V2G3-A09-52]	P	P	P	P	P	P	P	P	Only a part of the message sequence will be considered, so the requirement can only be tested partially Group_[V2G3-A09-52]
[V2G3-A09-53]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_CMN_VTB_CmValidate_010
[V2G3-A09-54]	P	P	P	P	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Table A.5 can only be tested partially Group_[V2G3-M09-12]_[V2G3-A09-54]
[V2G3-A09-56]	I	I	I	I	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-57] or [V2G3-A09-58] Group_[V2G3-A09-56]
[V2G3-A09-57]	—	—	—	—	X	X	X	X	Requirement is only testable if a corresponding PIXIT is considered Group_[V2G3-M09-07]_[V2G3-A09-57]
[V2G3-A09-58]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially TC_SECC_CMN_VTB_CmValidate_001 TC_SECC_CMN_VTB_CmValidate_009 TC_SECC_CMN_VTB_CmValidate_010 TC_SECC_CMN_VTB_CmValidate_011 TC_SECC_CMN_VTB_CmValidate_012
[V2G3-A09-59]	I	I	I	I	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-60] or [V2G3-A09-61] Group_[V2G3-A09-59]
[V2G3-A09-60]	—	—	—	—	X	X	X	X	Requirement is only testable if a corresponding PIXIT is considered Group_[V2G3-A09-60]
[V2G3-A09-61]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially TC_SECC_CMN_VTB_CmValidate_001 TC_SECC_CMN_VTB_CmValidate_011 TC_SECC_CMN_VTB_CmValidate_012
[V2G3-A09-62]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in EVCC black box configuration,

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmValidate_004 TC_EVCC_CMN_VTB_CmValidate_005 TC_EVCC_CMN_VTB_CmValidate_006 TC_EVCC_CMN_VTB_CmValidate_007
[V2G3-A09-63]	—	—	—	—	X	X	X	X	Requirement is only testable if a corresponding PIXIT is considered TC_EVCC_CMN_VTB_CmValidate_007
[V2G3-A09-64]	—	—	—	—	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-62] TC_EVCC_CMN_VTB_CmValidate_005 TC_EVCC_CMN_VTB_CmValidate_006
[V2G3-A09-65]	—	—	—	—	X	X	X	X	Requirement is only testable if a corresponding PIXIT is considered TC_EVCC_CMN_VTB_CmValidate_008
[V2G3-A09-66]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmValidate_018 TC_EVCC_CMN_VTB_CmValidate_019
[V2G3-A09-67]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmValidate_001 TC_EVCC_CMN_VTB_CmValidate_002 TC_EVCC_CMN_VTB_CmValidate_003
[V2G3-A09-68]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmValidate_001 TC_EVCC_CMN_VTB_CmValidate_002 TC_EVCC_CMN_VTB_CmValidate_003
[V2G3-A09-70]	—	—	—	—	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-71] TC_EVCC_CMN_VTB_CmValidate_009
[V2G3-A09-71]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmValidate_009
[V2G3-A09-72]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									TC_EVCC_CMN_VTB_CmValidate_010 TC_EVCC_CMN_VTB_CmValidate_011 TC_EVCC_CMN_VTB_CmValidate_012
[V2G3-A09-73]	—	—	—	—	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-74] TC_EVCC_CMN_VTB_CmSlacMatch_001
[V2G3-A09-74]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmSlacMatch_001
[V2G3-A09-75]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-75]
[V2G3-A09-76]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-82] TC_SECC_CMN_VTB_CmValidate_004 TC_SECC_CMN_VTB_CmValidate_005 TC_SECC_CMN_VTB_CmValidate_006 TC_SECC_CMN_VTB_CmValidate_007 TC_SECC_CMN_VTB_CmValidate_008
[V2G3-A09-77]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_CMN_VTB_CmValidate_002
[V2G3-A09-78]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_CMN_VTB_CmValidate_009
[V2G3-A09-79]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-75] Group_[V2G3-A09-79]
[V2G3-A09-80]	X	X	X	X	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered TC_SECC_CMN_VTB_CmValidate_010
[V2G3-A09-81]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Only one case can be considered, so the requirement can only be tested partially TC_SECC_CMN_VTB_CmValidate_011 TC_SECC_CMN_VTB_CmValidate_012
[V2G3-A09-82]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially TC_SECC_CMN_VTB_CmValidate_003

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									TC_SECC_CMN_VTB_CmValidate_004
[V2G3-A09-83]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-83] TC_SECC_CMN_VTB_CmValidate_003 TC_SECC_CMN_VTB_CmValidate_004
[V2G3-A09-84]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-84] TC_SECC_CMN_VTB_CmValidate_005 TC_SECC_CMN_VTB_CmValidate_006 TC_SECC_CMN_VTB_CmValidate_007 TC_SECC_CMN_VTB_CmValidate_008
[V2G3-A09-85]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-87] TC_SECC_CMN_VTB_CmValidate_001 TC_SECC_CMN_VTB_CmValidate_009 TC_SECC_CMN_VTB_CmValidate_011 TC_SECC_CMN_VTB_CmValidate_012
[V2G3-A09-86]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-87] TC_SECC_CMN_VTB_CmValidate_001 TC_SECC_CMN_VTB_CmValidate_009 TC_SECC_CMN_VTB_CmValidate_011 TC_SECC_CMN_VTB_CmValidate_012
[V2G3-A09-87]	P	P	P	P	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially TC_SECC_CMN_VTB_CmValidate_001 TC_SECC_CMN_VTB_CmValidate_009 TC_SECC_CMN_VTB_CmValidate_011 TC_SECC_CMN_VTB_CmValidate_012
[V2G3-A09-89]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-90] TC_SECC_CMN_VTB_CmSlacMatch_006
[V2G3-A09-90]	I	I	I	I	—	—	—	—	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-90] TC_SECC_CMN_VTB_CmSlacMatch_006
[V2G3-A09-91]	P	P	P	P	P	P	P	P	Only a part of the message sequence will be considered, so the requirement can only be tested partially Group_[V2G3-A09-91]
[V2G3-A09-94]	—	—	—	—	P	P	P	P	Internal states and timings cannot be tested

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									explicitly in EVCC black box configuration, so the requirement can only be tested partially Group_[V2G3-A09-94]
[V2G3-A09-95]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G3-A09-94] Group_[V2G3-A09-95]
[V2G3-A09-96]	P	P	P	P	—	—	—	—	Only one case can be considered, so the requirement can only be tested partially Group_[V2G3-A09-96]
[V2G3-A09-97]	X	X	X	X	—	—	—	—	TC_SECC_CMN_VTB_CmSlacMatch_003 TC_SECC_CMN_VTB_CmSlacMatch_004
[V2G3-A09-98]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G3-A09-96] Group_[V2G3-A09-98]
[V2G3-A09-99]	P	P	P	P	—	—	—	—	Internal timings cannot be tested explicitly in SECC black box configuration, so the requirement can only be tested partially TC_SECC_CMN_VTB_CmSlacMatch_001 TC_SECC_CMN_VTB_CmSlacMatch_002 TC_SECC_CMN_VTB_CmSlacMatch_003 TC_SECC_CMN_VTB_CmSlacMatch_004
[V2G3-A09-101]	—	—	—	—	I	I	I	I	Tested indirectly by consideration of requirement [V2G2-136]. This requirement refers to ISO 15118-2 document. TC_EVCC_CMN_VTB_CmAmpMap_001 TC_EVCC_CMN_VTB_CmAmpMap_002 TC_EVCC_CMN_VTB_CmAmpMap_008 TC_EVCC_CMN_VTB_PLCLinkStatus_001 TC_EVCC_CMN_VTB_PLCLinkStatus_005
[V2G3-A09-105]	I	I	I	I	—	—	—	—	Tested indirectly by consideration of requirement [V2G2-147]. This requirement refers to ISO 15118-2 document. TC_SECC_CMN_VTB_PLCLinkStatus_001 TC_SECC_AC_VTB_PLCLinkStatus_001 TC_SECC_AC_VTB_PLCLinkStatus_009
[V2G3-A09-106]	P	P	P	P	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Only a part of the message sequence (messages between hosts) can be considered, so the requirement can only be tested partially. Group_[V2G3-A09-106]
[V2G3-A09-109]	P	P	P	P	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Table A.9 can only be tested partially (large number of different carrier settings may apply. Can only be tested with random samples). Group_[V2G3-A09-109]_[V2G3-A09-111]
[V2G3-A09-110]	P	P	P	P	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Table A.9 can only be tested

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									partially. Group_[V2G3-A09-110]
[V2G3-A09-111]	P	P	P	P	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in SECC/EVCC black box configuration, so the requirement can only be tested partially. Group_[V2G3-A09-109]_[V2G3-A09-111]
[V2G3-A09-112]	P	P	P	P	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in SECC/EVCC black box configuration, so the requirement can only be tested partially. TC_EVCC_CMN_VTB_CmAmpMap_003 TC_EVCC_CMN_VTB_CmAmpMap_004 TC_SECC_CMN_VTB_CmAmpMap_003 TC_SECC_CMN_VTB_CmAmpMap_004
[V2G3-A09-113]	I	I	I	I	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [[V2G3-A09-115]. TC_EVCC_CMN_VTB_CmAmpMap_005 TC_SECC_CMN_VTB_CmAmpMap_005
[V2G3-A09-114]	I	I	I	I	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [[V2G3-A09-110]. TC_EVCC_CMN_VTB_CmAmpMap_004 TC_SECC_CMN_VTB_CmAmpMap_004
[V2G3-A09-115]	P	P	P	P	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal timings cannot be tested explicitly in SECC/EVCC black box configuration, so the requirement can only be tested partially. Group_[V2G3-A09-115]
[V2G3-A09-116]	X	X	X	X	X	X	X	X	Requirement is only testable if a corresponding PIXIT is considered. TC_EVCC_CMN_VTB_CmAmpMap_006 TC_EVCC_CMN_VTB_CmAmpMap_007 TC_SECC_CMN_VTB_CmAmpMap_006 TC_SECC_CMN_VTB_CmAmpMap_007
[V2G3-A09-118]	P	P	P	P	N	N	N	N	Requirement definition considers various possible scenarios, so the requirement can only be tested partially for exemplarily selected cases for the profile defined in the respective column. Cannot be tested explicitly in EVCC black box test configuration for the profile defined in the respective column. TC_SECC_CMN_VTB_PLCLinkStatus_003
[V2G3-A09-121]	P	P	P	P	P	P	P	P	Internal states cannot be tested explicitly in SECC/EVCC black box configuration, so the requirement can only be tested partially

Requirement ID	Covered in Test Suite								TC ID(s)/Comment
	SECC				EVCC				
	EIM		PnC		EIM		PnC		
	AC	DC	AC	DC	AC	DC	AC	DC	
									TC_EVCC_AC_VTB_PLCLinkStatus_006 TC_EVCC_DC_VTB_PLCLinkStatus_003 TC_SECC_AC_VTB_PLCLinkStatus_008 TC_SECC_DC_VTB_PLCLinkStatus_005
[V2G3-A09-122]	—	—	—	—	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-123] TC_EVCC_CMN_VTB_CmSlacParm_010
[V2G3-A09-123]	—	—	—	—	X	X	X	X	Requirement is only testable if a corresponding PIXIT is considered TC_EVCC_CMN_VTB_CmSlacParm_010
[V2G3-A09-124]	—	—	—	—	I	I	I	I	Requirement is only testable if a corresponding PIXIT is considered. Tested indirectly by consideration of requirement [V2G3-A09-123] TC_EVCC_CMN_VTB_CmSlacParm_010
[V2G3-A09-125]	—	—	—	—	P	P	P	P	Requirement is only testable if a corresponding PIXIT is considered. Internal states cannot be tested explicitly in EVCC black box configuration, so the requirement can only be tested partially TC_EVCC_CMN_VTB_CmSlacParm_010
[V2G3-A09-126]	P	P	P	P	N	N	N	N	Requirement definition considers various possible scenarios, so the requirement can only be tested partially for exemplarily selected cases for the profile defined in the respective column. Cannot be tested explicitly in EVCC black box test configuration for the profile defined in the respective column TC_SECC_CMN_VTB_AttenuationCharacterization_019 TC_SECC_CMN_VTB_CmSlacMatch_023 TC_SECC_CMN_VTB_CmSlacMatch_024 TC_SECC_CMN_VTB_CmSlacParm_007 TC_SECC_CMN_VTB_CmValidate_013
[V2G3-A09-127]	N	N	N	N	P	P	P	P	Cannot be tested explicitly in SECC black box test configuration for the profile defined in the respective column. Requirement definition considers various possible scenarios, so the requirement can only be tested partially for exemplarily selected cases for the profile defined in the respective column. Group_[V2G3-A09-127]

Several requirements of ISO 15118-3 are covered in multiple test cases of the ATS. For a simplified representation, the relevant TC identifier of each test case is mapped to the corresponding requirement. These special groups are summarized in Table 30. If two requirement groups refer to an identical set of TC identifiers, these groups are consolidated.

Table 30 — Groups for a simplified TC Id representation (see Table 29)

Group name	TC IDs
Group_[V2G3-A09-18]	TC_EVCC_CMN_VTB_AttenuationCharacterization_001, TC_EVCC_CMN_VTB_AttenuationCharacterization_002,

Group name	TC IDs
	TC_EVCC_CMN_VTB_AttenuationCharacterization_003, TC_EVCC_CMN_VTB_AttenuationCharacterization_004, TC_EVCC_CMN_VTB_AttenuationCharacterization_005, TC_EVCC_CMN_VTB_AttenuationCharacterization_006, TC_EVCC_CMN_VTB_AttenuationCharacterization_007, TC_EVCC_CMN_VTB_AttenuationCharacterization_008, TC_EVCC_CMN_VTB_AttenuationCharacterization_009, TC_EVCC_CMN_VTB_AttenuationCharacterization_010, TC_EVCC_CMN_VTB_AttenuationCharacterization_011, TC_EVCC_AC_VTB_AttenuationCharacterization_001, TC_EVCC_AC_VTB_AttenuationCharacterization_002, TC_EVCC_CMN_VTB_CmSlacParm_001, TC_EVCC_CMN_VTB_CmSlacParm_002, TC_EVCC_CMN_VTB_CmSlacParm_003, TC_EVCC_CMN_VTB_CmSlacParm_004, TC_EVCC_CMN_VTB_CmSlacParm_005, TC_EVCC_CMN_VTB_CmSlacParm_006, TC_EVCC_CMN_VTB_CmSlacParm_007, TC_EVCC_CMN_VTB_CmSlacParm_008, TC_EVCC_CMN_VTB_CmSlacParm_010, TC_EVCC_CMN_VTB_CmSlacParm_011, TC_EVCC_CMN_VTB_CmSlacParm_012, TC_EVCC_CMN_VTB_CmSlacParm_013, TC_EVCC_CMN_VTB_CmSlacParm_014, TC_EVCC_AC_VTB_CmSlacParm_001, TC_EVCC_AC_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_AttenuationCharacterization_001, TC_SECC_CMN_VTB_AttenuationCharacterization_020, TC_SECC_CMN_VTB_CmSlacParm_001, TC_SECC_CMN_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_003, TC_SECC_CMN_VTB_CmSlacParm_004, TC_SECC_CMN_VTB_CmSlacParm_005, TC_SECC_CMN_VTB_CmSlacParm_006, TC_SECC_CMN_VTB_CmSlacParm_008, TC_SECC_CMN_VTB_CmSlacParm_009, TC_SECC_AC_VTB_CmSlacParm_001, TC_SECC_AC_VTB_CmSlacParm_002, TC_SECC_AC_VTB_CmSlacParm_003, TC_SECC_AC_VTB_CmSlacParm_004, TC_SECC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-A09-23]	TC_EVCC_CMN_VTB_AttenuationCharacterization_001, TC_EVCC_CMN_VTB_AttenuationCharacterization_002, TC_EVCC_CMN_VTB_AttenuationCharacterization_003, TC_EVCC_CMN_VTB_AttenuationCharacterization_004, TC_EVCC_CMN_VTB_AttenuationCharacterization_005, TC_EVCC_CMN_VTB_AttenuationCharacterization_006, TC_EVCC_CMN_VTB_AttenuationCharacterization_007, TC_EVCC_CMN_VTB_AttenuationCharacterization_008, TC_EVCC_CMN_VTB_AttenuationCharacterization_009, TC_EVCC_CMN_VTB_AttenuationCharacterization_010, TC_EVCC_CMN_VTB_AttenuationCharacterization_011, TC_EVCC_AC_VTB_AttenuationCharacterization_001, TC_EVCC_AC_VTB_AttenuationCharacterization_002, TC_SECC_CMN_VTB_AttenuationCharacterization_001, TC_SECC_CMN_VTB_AttenuationCharacterization_002, TC_SECC_CMN_VTB_AttenuationCharacterization_003, TC_SECC_CMN_VTB_AttenuationCharacterization_004, TC_SECC_CMN_VTB_AttenuationCharacterization_005, TC_SECC_CMN_VTB_AttenuationCharacterization_006, TC_SECC_CMN_VTB_AttenuationCharacterization_007, TC_SECC_CMN_VTB_AttenuationCharacterization_008, TC_SECC_CMN_VTB_AttenuationCharacterization_009, TC_SECC_CMN_VTB_AttenuationCharacterization_010, TC_SECC_CMN_VTB_AttenuationCharacterization_011, TC_SECC_CMN_VTB_AttenuationCharacterization_012, TC_SECC_CMN_VTB_AttenuationCharacterization_013, TC_SECC_CMN_VTB_AttenuationCharacterization_014,

Group name	TC IDs
	TC_SECC_CMN_VTB_AttenuationCharacterization_015, TC_SECC_CMN_VTB_AttenuationCharacterization_016, TC_SECC_CMN_VTB_AttenuationCharacterization_017, TC_SECC_CMN_VTB_AttenuationCharacterization_018, TC_SECC_CMN_VTB_AttenuationCharacterization_020,
Group_[V2G3-A09-35]	TC_EVCC_CMN_VTB_AttenuationCharacterization_004, TC_EVCC_CMN_VTB_AttenuationCharacterization_005, TC_EVCC_CMN_VTB_AttenuationCharacterization_006, TC_EVCC_CMN_VTB_AttenuationCharacterization_007, TC_EVCC_CMN_VTB_AttenuationCharacterization_008, TC_EVCC_CMN_VTB_AttenuationCharacterization_009, TC_EVCC_CMN_VTB_AttenuationCharacterization_010, TC_EVCC_CMN_VTB_AttenuationCharacterization_011,
Group_[V2G3-A09-38]	TC_EVCC_CMN_VTB_CmSlacMatch_001, TC_EVCC_CMN_VTB_CmSlacMatch_002, TC_EVCC_CMN_VTB_CmSlacMatch_003, TC_EVCC_CMN_VTB_CmSlacMatch_004, TC_EVCC_CMN_VTB_CmSlacMatch_005, TC_EVCC_CMN_VTB_CmSlacMatch_006, TC_EVCC_CMN_VTB_CmSlacMatch_007, TC_EVCC_CMN_VTB_CmSlacMatch_008, TC_EVCC_CMN_VTB_CmSlacMatch_009, TC_EVCC_CMN_VTB_CmSlacMatch_010, TC_EVCC_CMN_VTB_CmSlacMatch_011, TC_EVCC_CMN_VTB_CmSlacMatch_012, TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001,
Group_[V2G3-A09-52]	TC_EVCC_CMN_VTB_CmSlacMatch_001, TC_EVCC_CMN_VTB_CmSlacMatch_002, TC_EVCC_CMN_VTB_CmSlacMatch_003, TC_EVCC_CMN_VTB_CmSlacMatch_004, TC_EVCC_CMN_VTB_CmSlacMatch_005, TC_EVCC_CMN_VTB_CmSlacMatch_006, TC_EVCC_CMN_VTB_CmSlacMatch_007, TC_EVCC_CMN_VTB_CmSlacMatch_008, TC_EVCC_CMN_VTB_CmSlacMatch_009, TC_EVCC_CMN_VTB_CmSlacMatch_010, TC_EVCC_CMN_VTB_CmSlacMatch_011, TC_EVCC_CMN_VTB_CmSlacMatch_012, TC_EVCC_CMN_VTB_CmValidate_001, TC_EVCC_CMN_VTB_CmValidate_002, TC_EVCC_CMN_VTB_CmValidate_003, TC_EVCC_CMN_VTB_CmValidate_004, TC_EVCC_CMN_VTB_CmValidate_005, TC_EVCC_CMN_VTB_CmValidate_006, TC_EVCC_CMN_VTB_CmValidate_007, TC_EVCC_CMN_VTB_CmValidate_008, TC_EVCC_CMN_VTB_CmValidate_009, TC_EVCC_CMN_VTB_CmValidate_010, TC_EVCC_CMN_VTB_CmValidate_011, TC_EVCC_CMN_VTB_CmValidate_012, TC_EVCC_CMN_VTB_CmValidate_013, TC_EVCC_CMN_VTB_CmValidate_014, TC_EVCC_CMN_VTB_CmValidate_015, TC_EVCC_CMN_VTB_CmValidate_016, TC_EVCC_CMN_VTB_CmValidate_017, TC_EVCC_CMN_VTB_CmValidate_018, TC_EVCC_CMN_VTB_CmValidate_019, TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001, TC_SECC_CMN_VTB_CmSlacMatch_001, TC_SECC_CMN_VTB_CmSlacMatch_002, TC_SECC_CMN_VTB_CmSlacMatch_003, TC_SECC_CMN_VTB_CmSlacMatch_004, TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012,
Group_[V2G3-A09-91]	TC_EVCC_CMN_VTB_CmSlacMatch_001, TC_EVCC_CMN_VTB_CmSlacMatch_002, TC_EVCC_CMN_VTB_CmSlacMatch_003, TC_EVCC_CMN_VTB_CmSlacMatch_004, TC_EVCC_CMN_VTB_CmSlacMatch_005, TC_EVCC_CMN_VTB_CmSlacMatch_006, TC_EVCC_CMN_VTB_CmSlacMatch_007, TC_EVCC_CMN_VTB_CmSlacMatch_008,

Group name	TC IDs
	TC_EVCC_CMN_VTB_CmSlacMatch_009, TC_EVCC_CMN_VTB_CmSlacMatch_010, TC_EVCC_CMN_VTB_CmSlacMatch_011, TC_EVCC_CMN_VTB_CmSlacMatch_012, TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001, TC_SECC_CMN_VTB_CmSlacMatch_001, TC_SECC_CMN_VTB_CmSlacMatch_002, TC_SECC_CMN_VTB_CmSlacMatch_003, TC_SECC_CMN_VTB_CmSlacMatch_004,
Group_[V2G3-A08-01]	TC_EVCC_CMN_VTB_CmSlacMatch_003, TC_EVCC_CMN_VTB_CmSlacMatch_004, TC_EVCC_CMN_VTB_CmSlacMatch_005, TC_EVCC_CMN_VTB_CmSlacMatch_006, TC_EVCC_CMN_VTB_CmSlacMatch_007, TC_EVCC_CMN_VTB_CmSlacMatch_008, TC_EVCC_CMN_VTB_CmSlacMatch_009, TC_EVCC_CMN_VTB_CmSlacMatch_010, TC_EVCC_CMN_VTB_CmSlacMatch_011, TC_EVCC_CMN_VTB_CmSlacParm_002, TC_EVCC_CMN_VTB_CmSlacParm_003, TC_EVCC_CMN_VTB_CmSlacParm_004, TC_EVCC_CMN_VTB_CmSlacParm_005, TC_EVCC_CMN_VTB_CmSlacParm_006, TC_EVCC_CMN_VTB_CmSlacParm_007, TC_EVCC_CMN_VTB_CmSlacParm_008, TC_EVCC_CMN_VTB_CmSlacParm_010, TC_EVCC_CMN_VTB_CmSlacParm_011, TC_EVCC_CMN_VTB_CmValidate_004, TC_EVCC_CMN_VTB_CmValidate_005, TC_EVCC_CMN_VTB_CmValidate_006, TC_EVCC_CMN_VTB_CmValidate_007, TC_EVCC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_AttenuationCharacterization_004, TC_SECC_CMN_VTB_AttenuationCharacterization_005, TC_SECC_CMN_VTB_AttenuationCharacterization_006, TC_SECC_CMN_VTB_AttenuationCharacterization_007, TC_SECC_CMN_VTB_AttenuationCharacterization_008, TC_SECC_CMN_VTB_AttenuationCharacterization_009, TC_SECC_CMN_VTB_AttenuationCharacterization_010, TC_SECC_CMN_VTB_AttenuationCharacterization_011, TC_SECC_CMN_VTB_CmSlacParm_004, TC_SECC_CMN_VTB_CmSlacParm_005, TC_SECC_CMN_VTB_CmSlacParm_006, TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_003, TC_SECC_CMN_VTB_CmValidate_004, TC_SECC_CMN_VTB_CmValidate_005, TC_SECC_CMN_VTB_CmValidate_006, TC_SECC_CMN_VTB_CmValidate_007, TC_SECC_CMN_VTB_CmValidate_008, TC_SECC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_CmValidate_010, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012,
Group_[V2G3-A09-94]	TC_EVCC_CMN_VTB_CmSlacMatch_003, TC_EVCC_CMN_VTB_CmSlacMatch_004, TC_EVCC_CMN_VTB_CmSlacMatch_005, TC_EVCC_CMN_VTB_CmSlacMatch_006, TC_EVCC_CMN_VTB_CmSlacMatch_007, TC_EVCC_CMN_VTB_CmSlacMatch_008, TC_EVCC_CMN_VTB_CmSlacMatch_009, TC_EVCC_CMN_VTB_CmSlacMatch_010, TC_EVCC_CMN_VTB_CmSlacMatch_011,
Group_[V2G3-A09-95]	TC_EVCC_CMN_VTB_CmSlacMatch_004, TC_EVCC_CMN_VTB_CmSlacMatch_005, TC_EVCC_CMN_VTB_CmSlacMatch_006, TC_EVCC_CMN_VTB_CmSlacMatch_007, TC_EVCC_CMN_VTB_CmSlacMatch_008, TC_EVCC_CMN_VTB_CmSlacMatch_009, TC_EVCC_CMN_VTB_CmSlacMatch_010, TC_EVCC_CMN_VTB_CmSlacMatch_011,

Group name	TC IDs
Group_[V2G3-A09-04]	TC_EVCC_CMN_VTB_CmSlacParm_001, TC_EVCC_CMN_VTB_CmSlacParm_002, TC_EVCC_CMN_VTB_CmSlacParm_003, TC_EVCC_CMN_VTB_CmSlacParm_004, TC_EVCC_CMN_VTB_CmSlacParm_005, TC_EVCC_CMN_VTB_CmSlacParm_006, TC_EVCC_CMN_VTB_CmSlacParm_007, TC_EVCC_CMN_VTB_CmSlacParm_008, TC_EVCC_CMN_VTB_CmSlacParm_009, TC_EVCC_CMN_VTB_CmSlacParm_010, TC_EVCC_CMN_VTB_CmSlacParm_011, TC_EVCC_CMN_VTB_CmSlacParm_012, TC_EVCC_CMN_VTB_CmSlacParm_013, TC_EVCC_CMN_VTB_CmSlacParm_014, TC_EVCC_AC_VTB_CmSlacParm_001, TC_EVCC_AC_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_001, TC_SECC_CMN_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_003, TC_SECC_CMN_VTB_CmSlacParm_008, TC_SECC_CMN_VTB_CmSlacParm_009, TC_SECC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-A09-09]	TC_EVCC_CMN_VTB_CmSlacParm_003, TC_EVCC_CMN_VTB_CmSlacParm_004, TC_EVCC_CMN_VTB_CmSlacParm_005, TC_EVCC_CMN_VTB_CmSlacParm_006, TC_EVCC_CMN_VTB_CmSlacParm_007, TC_EVCC_CMN_VTB_CmSlacParm_008, TC_EVCC_CMN_VTB_CmSlacParm_011,
Group_[V2G3-A09-56]	TC_EVCC_CMN_VTB_CmValidate_001, TC_EVCC_CMN_VTB_CmValidate_002, TC_EVCC_CMN_VTB_CmValidate_003, TC_EVCC_CMN_VTB_CmValidate_004, TC_EVCC_CMN_VTB_CmValidate_005, TC_EVCC_CMN_VTB_CmValidate_006, TC_EVCC_CMN_VTB_CmValidate_007, TC_EVCC_CMN_VTB_CmValidate_008, TC_EVCC_CMN_VTB_CmValidate_009, TC_EVCC_CMN_VTB_CmValidate_010, TC_EVCC_CMN_VTB_CmValidate_011, TC_EVCC_CMN_VTB_CmValidate_012, TC_EVCC_CMN_VTB_CmValidate_013, TC_EVCC_CMN_VTB_CmValidate_014, TC_EVCC_CMN_VTB_CmValidate_015, TC_EVCC_CMN_VTB_CmValidate_016, TC_EVCC_CMN_VTB_CmValidate_017, TC_EVCC_CMN_VTB_CmValidate_018, TC_EVCC_CMN_VTB_CmValidate_019, TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_CmValidate_010, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012,
Group_[V2G3-A09-59]	TC_EVCC_CMN_VTB_CmValidate_001, TC_EVCC_CMN_VTB_CmValidate_002, TC_EVCC_CMN_VTB_CmValidate_003, TC_EVCC_CMN_VTB_CmValidate_009, TC_EVCC_CMN_VTB_CmValidate_010, TC_EVCC_CMN_VTB_CmValidate_011, TC_EVCC_CMN_VTB_CmValidate_012, TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012,
Group_[V2G3-A09-60]	TC_EVCC_CMN_VTB_CmValidate_001, TC_EVCC_CMN_VTB_CmValidate_002, TC_EVCC_CMN_VTB_CmValidate_003, TC_EVCC_CMN_VTB_CmValidate_009, TC_EVCC_CMN_VTB_CmValidate_010, TC_EVCC_CMN_VTB_CmValidate_011, TC_EVCC_CMN_VTB_CmValidate_012,
Group_[V2G3-M07-03]	TC_EVCC_CMN_VTB_PLCLinkStatus_006, TC_EVCC_CMN_VTB_PLCLinkStatus_007, TC_EVCC_CMN_VTB_PLCLinkStatus_008, TC_EVCC_AC_VTB_PLCLinkStatus_003, TC_EVCC_AC_VTB_PLCLinkStatus_004, TC_EVCC_AC_VTB_PLCLinkStatus_005, TC_EVCC_AC_VTB_PLCLinkStatus_007, TC_EVCC_AC_VTB_PLCLinkStatus_008, TC_SECC_CMN_VTB_PLCLinkStatus_005, TC_SECC_AC_VTB_PLCLinkStatus_005, TC_SECC_AC_VTB_PLCLinkStatus_006, TC_SECC_AC_VTB_PLCLinkStatus_007, TC_SECC_AC_VTB_PLCLinkStatus_011, TC_SECC_DC_VTB_PLCLinkStatus_004,
Group_[V2G3-M12-01]	TC_EVCC_CMN_VTB_PLCLinkStatus_008, TC_EVCC_AC_VTB_PLCLinkStatus_001,

Group name	TC IDs
	TC_EVCC_AC_VTB_PLCLinkStatus_002, TC_EVCC_AC_VTB_PLCLinkStatus_009, TC_EVCC_AC_VTB_PLCLinkStatus_010, TC_EVCC_DC_VTB_PLCLinkStatus_001, TC_EVCC_DC_VTB_PLCLinkStatus_002, TC_EVCC_DC_VTB_PLCLinkStatus_004, TC_EVCC_DC_VTB_PLCLinkStatus_005, TC_SECC_CMN_VTB_PLCLinkStatus_005, TC_SECC_AC_VTB_PLCLinkStatus_002, TC_SECC_AC_VTB_PLCLinkStatus_003, TC_SECC_AC_VTB_PLCLinkStatus_010, TC_SECC_AC_VTB_PLCLinkStatus_012, TC_SECC_DC_VTB_PLCLinkStatus_001, TC_SECC_DC_VTB_PLCLinkStatus_002, TC_SECC_DC_VTB_PLCLinkStatus_006, TC_SECC_DC_VTB_PLCLinkStatus_007,
Group_[V2G3-M07-21]	TC_EVCC_AC_VTB_PLCLinkStatus_001, TC_EVCC_AC_VTB_PLCLinkStatus_002, TC_EVCC_AC_VTB_PLCLinkStatus_009, TC_EVCC_AC_VTB_PLCLinkStatus_010, TC_EVCC_DC_VTB_PLCLinkStatus_001, TC_EVCC_DC_VTB_PLCLinkStatus_002, TC_EVCC_DC_VTB_PLCLinkStatus_004, TC_EVCC_DC_VTB_PLCLinkStatus_005, TC_SECC_AC_VTB_PLCLinkStatus_002, TC_SECC_DC_VTB_PLCLinkStatus_001,
Group_[V2G3-M09-16]	TC_EVCC_AC_VTB_PLCLinkStatus_001, TC_EVCC_AC_VTB_PLCLinkStatus_002, TC_EVCC_AC_VTB_PLCLinkStatus_009, TC_EVCC_AC_VTB_PLCLinkStatus_010, TC_EVCC_DC_VTB_PLCLinkStatus_001, TC_EVCC_DC_VTB_PLCLinkStatus_004, TC_EVCC_DC_VTB_PLCLinkStatus_005, TC_SECC_AC_VTB_PLCLinkStatus_003, TC_SECC_AC_VTB_PLCLinkStatus_010, TC_SECC_AC_VTB_PLCLinkStatus_012, TC_SECC_DC_VTB_PLCLinkStatus_002, TC_SECC_DC_VTB_PLCLinkStatus_006, TC_SECC_DC_VTB_PLCLinkStatus_007,
Group_[V2G3-M07-28]	TC_EVCC_AC_VTB_PLCLinkStatus_002, TC_EVCC_AC_VTB_PLCLinkStatus_009, TC_EVCC_AC_VTB_PLCLinkStatus_010, TC_EVCC_DC_VTB_PLCLinkStatus_002, TC_EVCC_DC_VTB_PLCLinkStatus_004, TC_EVCC_DC_VTB_PLCLinkStatus_005,
Group_[V2G3-A09-46]	TC_SECC_CMN_VTB_AttenuationCharacterization_004, TC_SECC_CMN_VTB_AttenuationCharacterization_005, TC_SECC_CMN_VTB_AttenuationCharacterization_006, TC_SECC_CMN_VTB_AttenuationCharacterization_007, TC_SECC_CMN_VTB_AttenuationCharacterization_008, TC_SECC_CMN_VTB_AttenuationCharacterization_009, TC_SECC_CMN_VTB_AttenuationCharacterization_010, TC_SECC_CMN_VTB_AttenuationCharacterization_011,
Group_[V2G3-A09-47]	TC_SECC_CMN_VTB_AttenuationCharacterization_005, TC_SECC_CMN_VTB_AttenuationCharacterization_006, TC_SECC_CMN_VTB_AttenuationCharacterization_007, TC_SECC_CMN_VTB_AttenuationCharacterization_008, TC_SECC_CMN_VTB_AttenuationCharacterization_009, TC_SECC_CMN_VTB_AttenuationCharacterization_010, TC_SECC_CMN_VTB_AttenuationCharacterization_011,
Group_[V2G3-A09-41]	TC_SECC_CMN_VTB_AttenuationCharacterization_013, TC_SECC_CMN_VTB_AttenuationCharacterization_014, TC_SECC_CMN_VTB_AttenuationCharacterization_015, TC_SECC_CMN_VTB_AttenuationCharacterization_016, TC_SECC_CMN_VTB_AttenuationCharacterization_017, TC_SECC_CMN_VTB_AttenuationCharacterization_018,
Group_[V2G3-A09-96]	TC_SECC_CMN_VTB_CmSlacMatch_005, TC_SECC_CMN_VTB_CmSlacMatch_007,

Group name	TC IDs
	TC_SECC_CMN_VTB_CmSlacMatch_008, TC_SECC_CMN_VTB_CmSlacMatch_009, TC_SECC_CMN_VTB_CmSlacMatch_010, TC_SECC_CMN_VTB_CmSlacMatch_011, TC_SECC_CMN_VTB_CmSlacMatch_012, TC_SECC_CMN_VTB_CmSlacMatch_013, TC_SECC_CMN_VTB_CmSlacMatch_014, TC_SECC_CMN_VTB_CmSlacMatch_015, TC_SECC_CMN_VTB_CmSlacMatch_016, TC_SECC_CMN_VTB_CmSlacMatch_017, TC_SECC_CMN_VTB_CmSlacMatch_018, TC_SECC_CMN_VTB_CmSlacMatch_019, TC_SECC_CMN_VTB_CmSlacMatch_020, TC_SECC_CMN_VTB_CmSlacMatch_021, TC_SECC_CMN_VTB_CmSlacMatch_022,
Group_[V2G3-A09-98]	TC_SECC_CMN_VTB_CmSlacMatch_007, TC_SECC_CMN_VTB_CmSlacMatch_008, TC_SECC_CMN_VTB_CmSlacMatch_009, TC_SECC_CMN_VTB_CmSlacMatch_010, TC_SECC_CMN_VTB_CmSlacMatch_011, TC_SECC_CMN_VTB_CmSlacMatch_012, TC_SECC_CMN_VTB_CmSlacMatch_013, TC_SECC_CMN_VTB_CmSlacMatch_014, TC_SECC_CMN_VTB_CmSlacMatch_015, TC_SECC_CMN_VTB_CmSlacMatch_016, TC_SECC_CMN_VTB_CmSlacMatch_017, TC_SECC_CMN_VTB_CmSlacMatch_018, TC_SECC_CMN_VTB_CmSlacMatch_019, TC_SECC_CMN_VTB_CmSlacMatch_020, TC_SECC_CMN_VTB_CmSlacMatch_021, TC_SECC_CMN_VTB_CmSlacMatch_022,
Group_[V2G3-M06-11]	TC_SECC_CMN_VTB_CmSlacParm_001, TC_SECC_CMN_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_003, TC_SECC_CMN_VTB_CmSlacParm_004, TC_SECC_CMN_VTB_CmSlacParm_005, TC_SECC_CMN_VTB_CmSlacParm_006, TC_SECC_CMN_VTB_CmSlacParm_008, TC_SECC_CMN_VTB_CmSlacParm_009, TC_SECC_AC_VTB_CmSlacParm_001, TC_SECC_AC_VTB_CmSlacParm_002, TC_SECC_AC_VTB_CmSlacParm_003, TC_SECC_AC_VTB_CmSlacParm_004, TC_SECC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-A09-03]	TC_SECC_CMN_VTB_CmSlacParm_001, TC_SECC_CMN_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_003, TC_SECC_CMN_VTB_CmSlacParm_007, TC_SECC_CMN_VTB_CmSlacParm_008, TC_SECC_CMN_VTB_CmSlacParm_009, TC_SECC_CMN_VTB_PLCLinkStatus_003, TC_SECC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-M07-01]	TC_SECC_CMN_VTB_CmSlacParm_001, TC_SECC_CMN_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_003, TC_SECC_CMN_VTB_CmSlacParm_008, TC_SECC_CMN_VTB_CmSlacParm_009, TC_SECC_AC_VTB_CmSlacParm_001, TC_SECC_AC_VTB_CmSlacParm_002, TC_SECC_AC_VTB_CmSlacParm_003, TC_SECC_AC_VTB_CmSlacParm_004, TC_SECC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-A09-75]	TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_002, TC_SECC_CMN_VTB_CmValidate_003, TC_SECC_CMN_VTB_CmValidate_004, TC_SECC_CMN_VTB_CmValidate_005, TC_SECC_CMN_VTB_CmValidate_006, TC_SECC_CMN_VTB_CmValidate_007, TC_SECC_CMN_VTB_CmValidate_008, TC_SECC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_CmValidate_010, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012,

Group name	TC IDs
Group_[V2G3-A09-79]	TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_002, TC_SECC_CMN_VTB_CmValidate_003, TC_SECC_CMN_VTB_CmValidate_004, TC_SECC_CMN_VTB_CmValidate_005, TC_SECC_CMN_VTB_CmValidate_006, TC_SECC_CMN_VTB_CmValidate_007, TC_SECC_CMN_VTB_CmValidate_008, TC_SECC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012,
Group_[V2G3-M07-08]	TC_SECC_AC_VTB_CmSlacParm_001, TC_SECC_AC_VTB_CmSlacParm_002, TC_SECC_AC_VTB_CmSlacParm_003, TC_SECC_AC_VTB_PLCLinkStatus_004, TC_SECC_AC_VTB_PLCLinkStatus_005, TC_SECC_AC_VTB_PLCLinkStatus_006, TC_SECC_AC_VTB_PLCLinkStatus_011, TC_SECC_DC_VTB_PLCLinkStatus_003, TC_SECC_DC_VTB_PLCLinkStatus_004,
Group_[V2G3-M07-20]	TC_SECC_AC_VTB_PLCLinkStatus_002, TC_SECC_AC_VTB_PLCLinkStatus_003, TC_SECC_AC_VTB_PLCLinkStatus_010, TC_SECC_AC_VTB_PLCLinkStatus_012, TC_SECC_DC_VTB_PLCLinkStatus_001, TC_SECC_DC_VTB_PLCLinkStatus_002, TC_SECC_DC_VTB_PLCLinkStatus_006, TC_SECC_DC_VTB_PLCLinkStatus_007,
Group_[V2G3-A09-106]	TC_EVCC_CMN_VTB_CmAmpMap_001, TC_EVCC_CMN_VTB_CmAmpMap_002, TC_EVCC_CMN_VTB_CmAmpMap_003, TC_EVCC_CMN_VTB_CmAmpMap_004, TC_EVCC_CMN_VTB_CmAmpMap_005, TC_EVCC_CMN_VTB_CmAmpMap_006, TC_EVCC_CMN_VTB_CmAmpMap_007, TC_EVCC_CMN_VTB_CmAmpMap_008, TC_SECC_CMN_VTB_CmAmpMap_001, TC_SECC_CMN_VTB_CmAmpMap_002, TC_SECC_CMN_VTB_CmAmpMap_003, TC_SECC_CMN_VTB_CmAmpMap_004, TC_SECC_CMN_VTB_CmAmpMap_005, TC_SECC_CMN_VTB_CmAmpMap_006, TC_SECC_CMN_VTB_CmAmpMap_007, TC_SECC_CMN_VTB_CmAmpMap_008,
Group_[V2G3-A09-127]	TC_EVCC_CMN_VTB_AttenuationCharacterization_012, TC_EVCC_CMN_VTB_AttenuationCharacterization_013, TC_EVCC_CMN_VTB_CmValidate_020, TC_EVCC_CMN_VTB_CmValidate_021, TC_EVCC_CMN_VTB_PLCLinkStatus_003, TC_EVCC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-A09-115]	TC_EVCC_CMN_VTB_CmAmpMap_001, TC_EVCC_CMN_VTB_CmAmpMap_005, TC_EVCC_CMN_VTB_CmAmpMap_006, TC_EVCC_CMN_VTB_CmAmpMap_007, TC_EVCC_CMN_VTB_CmAmpMap_008, TC_SECC_CMN_VTB_CmAmpMap_001, TC_SECC_CMN_VTB_CmAmpMap_005, TC_SECC_CMN_VTB_CmAmpMap_006, TC_SECC_CMN_VTB_CmAmpMap_007, TC_SECC_CMN_VTB_CmAmpMap_008,
Group_[V2G3-A09-110]	TC_EVCC_CMN_VTB_CmAmpMap_001, TC_EVCC_CMN_VTB_CmAmpMap_006, TC_EVCC_CMN_VTB_CmAmpMap_007, TC_EVCC_CMN_VTB_CmAmpMap_008, TC_SECC_CMN_VTB_CmAmpMap_001, TC_SECC_CMN_VTB_CmAmpMap_006, TC_SECC_CMN_VTB_CmAmpMap_007, TC_SECC_CMN_VTB_CmAmpMap_008,
Group_[V2G3-A09-25]_[V2G3-A09-28]	TC_EVCC_CMN_VTB_AttenuationCharacterization_001, TC_EVCC_CMN_VTB_AttenuationCharacterization_002, TC_EVCC_CMN_VTB_AttenuationCharacterization_003, TC_EVCC_CMN_VTB_AttenuationCharacterization_004, TC_EVCC_CMN_VTB_AttenuationCharacterization_005, TC_EVCC_CMN_VTB_AttenuationCharacterization_006, TC_EVCC_CMN_VTB_AttenuationCharacterization_007, TC_EVCC_CMN_VTB_AttenuationCharacterization_008, TC_EVCC_CMN_VTB_AttenuationCharacterization_009, TC_EVCC_CMN_VTB_AttenuationCharacterization_010, TC_EVCC_CMN_VTB_AttenuationCharacterization_011, TC_EVCC_AC_VTB_AttenuationCharacterization_001, TC_EVCC_AC_VTB_AttenuationCharacterization_002,
Group_[V2G3-A09-01]_[V2G3-A09-17]	TC_EVCC_CMN_VTB_AttenuationCharacterization_001, TC_EVCC_CMN_VTB_AttenuationCharacterization_002, TC_EVCC_CMN_VTB_AttenuationCharacterization_003, TC_EVCC_CMN_VTB_AttenuationCharacterization_004, TC_EVCC_CMN_VTB_AttenuationCharacterization_005, TC_EVCC_CMN_VTB_AttenuationCharacterization_006, TC_EVCC_CMN_VTB_AttenuationCharacterization_007, TC_EVCC_CMN_VTB_AttenuationCharacterization_008, TC_EVCC_CMN_VTB_AttenuationCharacterization_009, TC_EVCC_CMN_VTB_AttenuationCharacterization_010,

Group name	TC IDs
	TC_EVCC_CMN_VTB_AttenuationCharacterization_011, TC_EVCC_AC_VTB_AttenuationCharacterization_001, TC_EVCC_AC_VTB_AttenuationCharacterization_002, TC_EVCC_CMN_VTB_CmAmpMap_001, TC_EVCC_CMN_VTB_CmAmpMap_002, TC_EVCC_CMN_VTB_CmAmpMap_008, TC_EVCC_CMN_VTB_CmSlacMatch_001, TC_EVCC_CMN_VTB_CmSlacMatch_002, TC_EVCC_CMN_VTB_CmSlacMatch_003, TC_EVCC_CMN_VTB_CmSlacMatch_004, TC_EVCC_CMN_VTB_CmSlacMatch_005, TC_EVCC_CMN_VTB_CmSlacMatch_006, TC_EVCC_CMN_VTB_CmSlacMatch_007, TC_EVCC_CMN_VTB_CmSlacMatch_008, TC_EVCC_CMN_VTB_CmSlacMatch_009, TC_EVCC_CMN_VTB_CmSlacMatch_010, TC_EVCC_CMN_VTB_CmSlacMatch_011, TC_EVCC_CMN_VTB_CmSlacMatch_012, TC_EVCC_CMN_VTB_CmSlacParm_001, TC_EVCC_CMN_VTB_CmSlacParm_002, TC_EVCC_CMN_VTB_CmSlacParm_003, TC_EVCC_CMN_VTB_CmSlacParm_004, TC_EVCC_CMN_VTB_CmSlacParm_005, TC_EVCC_CMN_VTB_CmSlacParm_006, TC_EVCC_CMN_VTB_CmSlacParm_007, TC_EVCC_CMN_VTB_CmSlacParm_008, TC_EVCC_CMN_VTB_CmSlacParm_009, TC_EVCC_CMN_VTB_CmSlacParm_010, TC_EVCC_CMN_VTB_CmSlacParm_011, TC_EVCC_CMN_VTB_CmSlacParm_012, TC_EVCC_CMN_VTB_CmSlacParm_013, TC_EVCC_CMN_VTB_CmSlacParm_014, TC_EVCC_AC_VTB_CmSlacParm_001, TC_EVCC_AC_VTB_CmSlacParm_002, TC_EVCC_CMN_VTB_CmValidate_001, TC_EVCC_CMN_VTB_CmValidate_002, TC_EVCC_CMN_VTB_CmValidate_003, TC_EVCC_CMN_VTB_CmValidate_004, TC_EVCC_CMN_VTB_CmValidate_005, TC_EVCC_CMN_VTB_CmValidate_006, TC_EVCC_CMN_VTB_CmValidate_007, TC_EVCC_CMN_VTB_CmValidate_008, TC_EVCC_CMN_VTB_CmValidate_009, TC_EVCC_CMN_VTB_CmValidate_010, TC_EVCC_CMN_VTB_CmValidate_011, TC_EVCC_CMN_VTB_CmValidate_012, TC_EVCC_CMN_VTB_CmValidate_013, TC_EVCC_CMN_VTB_CmValidate_014, TC_EVCC_CMN_VTB_CmValidate_015, TC_EVCC_CMN_VTB_CmValidate_016, TC_EVCC_CMN_VTB_CmValidate_017, TC_EVCC_CMN_VTB_CmValidate_018, TC_EVCC_CMN_VTB_CmValidate_019, TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001, TC_SECC_CMN_VTB_AttenuationCharacterization_001, TC_SECC_CMN_VTB_AttenuationCharacterization_002, TC_SECC_CMN_VTB_AttenuationCharacterization_003, TC_SECC_CMN_VTB_AttenuationCharacterization_004, TC_SECC_CMN_VTB_AttenuationCharacterization_005, TC_SECC_CMN_VTB_AttenuationCharacterization_006, TC_SECC_CMN_VTB_AttenuationCharacterization_007, TC_SECC_CMN_VTB_AttenuationCharacterization_008, TC_SECC_CMN_VTB_AttenuationCharacterization_009, TC_SECC_CMN_VTB_AttenuationCharacterization_010, TC_SECC_CMN_VTB_AttenuationCharacterization_011, TC_SECC_CMN_VTB_AttenuationCharacterization_020, TC_SECC_CMN_VTB_CmAmpMap_001, TC_SECC_CMN_VTB_CmAmpMap_002, TC_SECC_CMN_VTB_CmAmpMap_008, TC_SECC_CMN_VTB_CmSlacMatch_001, TC_SECC_CMN_VTB_CmSlacMatch_002, TC_SECC_CMN_VTB_CmSlacMatch_003, TC_SECC_CMN_VTB_CmSlacMatch_004, TC_SECC_CMN_VTB_CmSlacParm_001, TC_SECC_CMN_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_003, TC_SECC_CMN_VTB_CmSlacParm_008, TC_SECC_CMN_VTB_CmSlacParm_009, TC_SECC_AC_VTB_CmSlacParm_001, TC_SECC_AC_VTB_CmSlacParm_002, TC_SECC_AC_VTB_CmSlacParm_003,

Group name	TC IDs
	TC_SECC_AC_VTB_CmSlacParm_004, TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_002, TC_SECC_CMN_VTB_CmValidate_003, TC_SECC_CMN_VTB_CmValidate_004, TC_SECC_CMN_VTB_CmValidate_005, TC_SECC_CMN_VTB_CmValidate_006, TC_SECC_CMN_VTB_CmValidate_007, TC_SECC_CMN_VTB_CmValidate_008, TC_SECC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_CmValidate_010, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012, TC_SECC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-A09-05]_[V2G3-M06-13]	TC_EVCC_CMN_VTB_CmSlacParm_001, TC_EVCC_CMN_VTB_CmSlacParm_002, TC_EVCC_CMN_VTB_CmSlacParm_003, TC_EVCC_CMN_VTB_CmSlacParm_004, TC_EVCC_CMN_VTB_CmSlacParm_005, TC_EVCC_CMN_VTB_CmSlacParm_006, TC_EVCC_CMN_VTB_CmSlacParm_007, TC_EVCC_CMN_VTB_CmSlacParm_008, TC_EVCC_CMN_VTB_CmSlacParm_010, TC_EVCC_CMN_VTB_CmSlacParm_011, TC_EVCC_CMN_VTB_CmSlacParm_012, TC_EVCC_CMN_VTB_CmSlacParm_013, TC_EVCC_CMN_VTB_CmSlacParm_014, TC_EVCC_AC_VTB_CmSlacParm_001, TC_EVCC_AC_VTB_CmSlacParm_002,
Group_[V2G3-M09-07]_[V2G3-A09-57]	TC_EVCC_CMN_VTB_CmValidate_001, TC_EVCC_CMN_VTB_CmValidate_002, TC_EVCC_CMN_VTB_CmValidate_003, TC_EVCC_CMN_VTB_CmValidate_004, TC_EVCC_CMN_VTB_CmValidate_005, TC_EVCC_CMN_VTB_CmValidate_006, TC_EVCC_CMN_VTB_CmValidate_007, TC_EVCC_CMN_VTB_CmValidate_008, TC_EVCC_CMN_VTB_CmValidate_009, TC_EVCC_CMN_VTB_CmValidate_010, TC_EVCC_CMN_VTB_CmValidate_011, TC_EVCC_CMN_VTB_CmValidate_012, TC_EVCC_CMN_VTB_CmValidate_013, TC_EVCC_CMN_VTB_CmValidate_014, TC_EVCC_CMN_VTB_CmValidate_015, TC_EVCC_CMN_VTB_CmValidate_016, TC_EVCC_CMN_VTB_CmValidate_017, TC_EVCC_CMN_VTB_CmValidate_018, TC_EVCC_CMN_VTB_CmValidate_019,
Group_[V2G3-M09-12]_[V2G3-A09-54]	TC_EVCC_CMN_VTB_CmValidate_001, TC_EVCC_CMN_VTB_CmValidate_002, TC_EVCC_CMN_VTB_CmValidate_003, TC_EVCC_CMN_VTB_CmValidate_004, TC_EVCC_CMN_VTB_CmValidate_005, TC_EVCC_CMN_VTB_CmValidate_006, TC_EVCC_CMN_VTB_CmValidate_007, TC_EVCC_CMN_VTB_CmValidate_008, TC_EVCC_CMN_VTB_CmValidate_009, TC_EVCC_CMN_VTB_CmValidate_010, TC_EVCC_CMN_VTB_CmValidate_011, TC_EVCC_CMN_VTB_CmValidate_012, TC_EVCC_CMN_VTB_CmValidate_013, TC_EVCC_CMN_VTB_CmValidate_014, TC_EVCC_CMN_VTB_CmValidate_015, TC_EVCC_CMN_VTB_CmValidate_016, TC_EVCC_CMN_VTB_CmValidate_017, TC_EVCC_CMN_VTB_CmValidate_018, TC_EVCC_CMN_VTB_CmValidate_019, TC_SECC_CMN_VTB_CmValidate_001, TC_SECC_CMN_VTB_CmValidate_002, TC_SECC_CMN_VTB_CmValidate_003, TC_SECC_CMN_VTB_CmValidate_004, TC_SECC_CMN_VTB_CmValidate_005, TC_SECC_CMN_VTB_CmValidate_006, TC_SECC_CMN_VTB_CmValidate_007, TC_SECC_CMN_VTB_CmValidate_008, TC_SECC_CMN_VTB_CmValidate_009, TC_SECC_CMN_VTB_CmValidate_010, TC_SECC_CMN_VTB_CmValidate_011, TC_SECC_CMN_VTB_CmValidate_012,
Group_[V2G3-M07-19]_[V2G3-M07-29]	TC_EVCC_AC_VTB_PLCLinkStatus_001, TC_EVCC_AC_VTB_PLCLinkStatus_002, TC_EVCC_AC_VTB_PLCLinkStatus_009, TC_EVCC_AC_VTB_PLCLinkStatus_010, TC_EVCC_DC_VTB_PLCLinkStatus_001, TC_EVCC_DC_VTB_PLCLinkStatus_002, TC_EVCC_DC_VTB_PLCLinkStatus_004, TC_EVCC_DC_VTB_PLCLinkStatus_005,
Group_[V2G3-A09-44]_[V2G3-A09-45]	TC_SECC_CMN_VTB_AttenuationCharacterization_001, TC_SECC_CMN_VTB_AttenuationCharacterization_003, TC_SECC_CMN_VTB_AttenuationCharacterization_004, TC_SECC_CMN_VTB_AttenuationCharacterization_005, TC_SECC_CMN_VTB_AttenuationCharacterization_006, TC_SECC_CMN_VTB_AttenuationCharacterization_007,

Group name	TC IDs
	TC_SECC_CMN_VTB_AttenuationCharacterization_008, TC_SECC_CMN_VTB_AttenuationCharacterization_009, TC_SECC_CMN_VTB_AttenuationCharacterization_010, TC_SECC_CMN_VTB_AttenuationCharacterization_011, TC_SECC_CMN_VTB_AttenuationCharacterization_020,
Group_[V2G3-A09-42]_[V2G3-A09-43]	TC_SECC_CMN_VTB_AttenuationCharacterization_003, TC_SECC_CMN_VTB_AttenuationCharacterization_004, TC_SECC_CMN_VTB_AttenuationCharacterization_005, TC_SECC_CMN_VTB_AttenuationCharacterization_006, TC_SECC_CMN_VTB_AttenuationCharacterization_007, TC_SECC_CMN_VTB_AttenuationCharacterization_008, TC_SECC_CMN_VTB_AttenuationCharacterization_009, TC_SECC_CMN_VTB_AttenuationCharacterization_010, TC_SECC_CMN_VTB_AttenuationCharacterization_011,
Group_[V2G3-A09-11]_[V2G3-A09-15]	TC_SECC_CMN_VTB_CmSlacParm_001, TC_SECC_CMN_VTB_CmSlacParm_002, TC_SECC_CMN_VTB_CmSlacParm_003, TC_SECC_CMN_VTB_CmSlacParm_008, TC_SECC_CMN_VTB_CmSlacParm_009, TC_SECC_CMN_VTB_PLCLinkStatus_004,
Group_[V2G3-M07-31]_[V2G3-M07-32]	TC_SECC_AC_VTB_PLCLinkStatus_002, TC_SECC_AC_VTB_PLCLinkStatus_003, TC_SECC_AC_VTB_PLCLinkStatus_004, TC_SECC_AC_VTB_PLCLinkStatus_010, TC_SECC_AC_VTB_PLCLinkStatus_012, TC_SECC_DC_VTB_PLCLinkStatus_001, TC_SECC_DC_VTB_PLCLinkStatus_002, TC_SECC_DC_VTB_PLCLinkStatus_003, TC_SECC_DC_VTB_PLCLinkStatus_006, TC_SECC_DC_VTB_PLCLinkStatus_007,
Group_[V2G3-A09-109]_[V2G3-A09-111]	TC_EVCC_CMN_VTB_CmAmpMap_002, TC_EVCC_CMN_VTB_CmAmpMap_003, TC_EVCC_CMN_VTB_CmAmpMap_004, TC_SECC_CMN_VTB_CmAmpMap_002, TC_SECC_CMN_VTB_CmAmpMap_003, TC_SECC_CMN_VTB_CmAmpMap_004,
Group_[V2G3-A09-30]_[V2G3-A09-31]_[V2G3-A09-32]	TC_EVCC_CMN_VTB_AttenuationCharacterization_003, TC_EVCC_CMN_VTB_AttenuationCharacterization_004, TC_EVCC_CMN_VTB_AttenuationCharacterization_005, TC_EVCC_CMN_VTB_AttenuationCharacterization_006, TC_EVCC_CMN_VTB_AttenuationCharacterization_007, TC_EVCC_CMN_VTB_AttenuationCharacterization_008, TC_EVCC_CMN_VTB_AttenuationCharacterization_009, TC_EVCC_CMN_VTB_AttenuationCharacterization_010, TC_EVCC_CMN_VTB_AttenuationCharacterization_011,
Group_[V2G3-A09-07]_[V2G3-A09-08]_[V2G3-A09-10]	TC_EVCC_CMN_VTB_CmSlacParm_002, TC_EVCC_CMN_VTB_CmSlacParm_003, TC_EVCC_CMN_VTB_CmSlacParm_004, TC_EVCC_CMN_VTB_CmSlacParm_005, TC_EVCC_CMN_VTB_CmSlacParm_006, TC_EVCC_CMN_VTB_CmSlacParm_007, TC_EVCC_CMN_VTB_CmSlacParm_008, TC_EVCC_CMN_VTB_CmSlacParm_010, TC_EVCC_CMN_VTB_CmSlacParm_011,

7.6 Test case description

The test case descriptions in this document are described according to the template shown in Table 31.

Table 31 — Test case description template

TC Id	The TC Id is a unique identifier for a test case. It is specified according to the TC Id naming convention defined in 7.4.2.
Test objective	Short description of test objective according to the requirements from the base standard (15118-3).
Document reference	The document reference indicates the subclauses of the reference standard specifications in which the conformance requirement(s) is/are expressed. The references are provided according to the following format:

	Document: ISO 15118-X:20XX:(IS|FDIS) Section(s): x.x.x.x.x, y.y.y.y.y, …
Referenced requirement(s)	The referenced requirement(s) refers to the subclauses of the referenced standard specification requirement(s). The requirements are referenced according to the format defined in ISO 15118-3 or ISO 15118-2: [V2G3-YXX-ZZZ], …
Config Id	The Config Id references the ISO 15118-3 configuration selected for this test case according to 7.3.1. Example: CF_05_001, …
PICS selection	The PICS selection references the PICS statement(s) for this test case in accordance with 7.3.3. Example: PICS_CMN_CMN_IdentificationMode := eIM
PIXIT selection	The PIXIT selection references the PIXIT statement(s) for this test case in accordance with 7.3.4. Example: PIXIT_CMN_CMN_CmAmpMap := true
Pre-condition	
The pre-condition defines which state the SUT has to be before executing the actual Test Case. In the corresponding Test Case, when the execution of the initial condition does not succeed, it leads to the assignment of a “fail” verdict. The pre-condition is defined as reference to the pre-condition function in the ATS. Example: `f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
Definition of the events, which are parts of the test case objective, and the SUT are expected to perform in order to conform to the base specification. The expected behavior is defined as reference to the testbehavior function in the ATS. Example: `f_SECC_CMN_TB_VTB_CmSlacParm_001`	

7.7 Test case specification

The test case specifications are provided in the form of a TTCN-3 test suite according to the TTCN-3 Core Language as defined in ETSI ES 201 873-1 V4.6.1. This subclause defines additional conventions for the scope of this test suite.

7.7.1 Data types

7.7.1.1 SLAC

The TTCN-3 data structures for testing SLAC conformance are defined in G.3. The TTCN-3 data representation for SLAC requests and responses is defined according to ISO 15118-3:2015, A.9 and HomePlug GreenPHY Spec., release version 1.1.1, July 4, 2013.

7.7.2 Templates

A template defines at least one value for a data type. The templates in this document are used for two purposes:

— representation of stimulus test data; and

— SUT response matching with expected behavior.

The templates relevant for this document are defined in Annex F. Templates that are only used once in a concrete test case are defined in their corresponding context in the Annexes. All templates that are used in several test cases / contexts are defined in the common sections in the Annexes.

The templates for the SLAC messages consist of a header (MME_Header) and a payload (MME_Payload). There are templates that can be parameterized. At least one template for each message exists, where all elements of the first level of the message type are defined as parameters of the template.

In the TSS the following rules apply for templates used in the context of test cases for EVCC / SECC:

— For testing an SECC every attribute in an EVCC request message (test stimulus) is either allocated or omitted. In the SECC response message (SUT reaction) every attribute is allocated either by parameters of the template or TTCN-3 wildcards.

— For testing an EVCC every attribute in an SECC response message (test stimulus) is either allocated or omitted. In the EVCC request message (SUT reaction) every attribute is allocated either by parameters of the template or TTCN-3 wildcards.

7.7.3 Timeouts and timers

All timeouts are defined according to ISO 15118-3. For the scope of this document, *performance times* in ISO 15118-3 are irrelevant since they are not visible to the tester (black box testing). Hence only *timeouts* can be exactly measured from the tester's perspective and are considered in this document. The resulting timeout definitions for the TTCN-3 test suite are provided in A.1.

The timer handling in the test suite is defined as follows: Before sending a stimulus to the SUT the corresponding message timer is started. After receiving the expected response from the SUT the message timer will be stopped. In case no response is received from the SUT until the point in time defined by the corresponding timeout value, the test verdict is set to 'false'. The timer handling is modelled as part of the test behavior section within the test case specification.

7.7.4 Library functions

Within the TSS a few general utility functions are defined which may also be used in different contexts. Such functions are defined in D.4.

7.7.5 Test case modelling

One test case always executes a specific test behavior and verifies the SUT behavior according to its response. All test cases always begin and end in a defined and recoverable state of the SUT and the tester. As a consequence every test case consists of the following phases:

a) Initialize

   1) create PTC components; and

   2) map all MTC/PTC/system ports and communication bindings according to the underlying test configuration.

b) Pre-condition

   The test system is supposed to start in a state so that a communication or data exchange with the SUT is possible. In the pre-condition of a test case the tester as well as the SUT therefore initialize into known and stable state. This incorporates the following action:

   1) Bring SUT and tester to a valid and known state before the actual test behavior is executed.

c) Test behavior

   The actual test behavior defines the set of steps during the test execution which are essential in order to achieve the test purpose and assign verdicts to the possible outcomes. This involves the following actions:

   1) Initialize and start relevant timer(s) on the tester side;

2) Execute test behavior (send stimulus to the SUT);

3) Listen to any events and verify SUT response (stop timers); and

4) Assign test verdict for this test behavior.

d) Post-condition

1) Bring SUT and tester into final/initial state.

e) Shutdown

1) Unmap all connected ports and communication bindings; and

2) Shutdown PTC components.

7.7.6 SLAC Message handling for different SUT types

In reference to the test case modelling as defined in the previous subclause, a stimulus is always sent from the tester to the SUT as part of the test behavior. Due to the strictly defined client, server roles assigned to the EVCC and SECC respectively, this result in the following two message handling paradigms depending on the type of the SUT:

1. If the SUT is an SECC + PLC bridge: The tester sends one or several SLAC requests as stimuli to the SUT and expects a SLAC response back from the SUT. If multiple responses are defined as SUT behavior, the tester shall be able to handle these in order to allow for valid verdict assignments.

2. If the SUT is an EVCC + PLC bridge: The tester sends a SLAC response as stimulus to the SUT and expects one or several SLAC requests according to the message pattern in the SLAC message sequence back from the SUT. If multiple requests are defined as SUT behavior, the tester shall be able to handle these in order to allow for valid verdict assignments.

7.7.7 IEC 61851-1 PWM event handling and control

The parallel test component (PTC) for IEC 61851-1 signaling continuously observes the protocol specific parameters positive voltage (state), frequency, duty cycle and the proximity resistor value in accordance with the defined valid range of values (as defined in IEC 61851-1:2017, Annex A), during the entire test case execution.

Requirements of IEC 61851-1 SUT adapter for SECC or EVCC testing

[V2G5-011] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a function to get the current PWM state or associated voltage level.

[V2G5-012] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a call-back function which reports change events of the current PWM state or associated voltage level.

[V2G5-013] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a function to set the proximity on SUT side.

[V2G5-014] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a function to get the proximity resistor value from SUT side.

[V2G5-015] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a call-back function which reports change events of the resistor value for the proximity from SUT side.

[V2G5-016] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a function to get the duty cycle in [%].

[V2G5-017] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a call-back function which reports changes of the duty cycle in [%].

[V2G5-018] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a function to get the frequency in [Hz].

[V2G5-019] The IEC 61851-1 SUT adapter for SECC or EVCC testing shall provide a call-back function which reports change events of the frequency in [Hz].

Requirements of IEC 61851-1 SUT adapter for SECC testing

[V2G5-020] The IEC 61851-1 SUT adapter for SECC testing shall provide a function to set the current PWM state or associated voltage level. State B, C and D or the associated voltage level shall be supported, other voltage level are optional.

Requirements of IEC 61851-1 SUT adapter for EVCC testing

[V2G5-021] The IEC 61851-1 SUT adapter for EVCC testing shall provide a function to set the current PWM state or associated voltage level. State E and F or the associated voltage level shall be supported, other voltage levels are optional.

[V2G5-022] The IEC 61851-1 SUT adapter for EVCC testing shall provide a function to set the current duty cycle in [%]. The values 5 %, and 7 %–98 % shall be supported, other values are optional.

[V2G5-023] The IEC 61851-1 SUT adapter for EVCC testing shall provide a function to set current frequency in [Hz]. A value of 1 000 Hz ± 5 % shall be supported, other values are optional.

[V2G5-024] The IEC 61851-1 SUT adapter for EVCC testing shall provide a function to switch on/off the ControlPilot Line to simulate an un-plug or plug-in for the SUT. State A or the associated voltage level shall be supported, other voltage level is optional.

Requirements of IEC 61851-1 PTC for SECC or EVCC testing

[V2G5-025] The IEC 61851 PTC for SECC or EVCC testing shall provide a function which activates the validity monitoring as specified in [V2G4-036], [V2G4-037], [V2G4-038], [V2G4-039], [V2G4-040], [V2G4-041].

[V2G5-026] The IEC 61851-1 PTC for SECC or EVCC testing shall provide a function to confirm a change from/to a given PWM state. This function shall wait for receiving a state change event from IEC 61851-1 SUT adapter. If no change event occurs within a given time [ms], the function shall return an error.

[V2G5-027] The IEC 61851-1 PTC for SECC or EVCC testing shall provide a function to change its valid PWM state condition. This function shall change the internal monitoring condition to a given PWM state, which the IEC 61851-1 PTC will consider as the next valid PWM state.

[V2G5-028] The IEC 61851-1 PTC for SECC or EVCC testing shall provide a function to change its valid duty cycle range. This function shall change the internal monitoring condition to a given duty cycle range from [%] – to [%], which the IEC 61851-1 PTC will consider as the next valid duty cycle range.

Requirements of IEC 61851-1 PTC for SECC testing

[V2G5-029] The IEC 61851-1 PTC for SECC testing shall provide a function to confirm a given duty cycle in [%]. This function shall wait for receiving a duty cycle change event from IEC 61851-1 SUT adapter. If no change event occurs within a given time [ms], the function shall report an error.

[V2G5-030] The IEC 61851-1 PTC for SECC testing shall provide a function to change its valid frequency range. This function shall change the internal monitoring condition to a given frequency range from [Hz] – to [Hz], which the IEC 61851-1 PTC will consider as the next frequency range.

7.7.8 Data link status control functionality

In order to ensure data link processing, the MTC shall be able to detect and control link status information by using special functions depending on the MTC's type (EVCC_Tester / SECC_Tester).

Requirements of data link status control for EVCC or SECC testing

[V2G5-031]	The service function for data link status control shall provide a mean to check if the PLC link can be established within TT_match_join.
[V2G5-032]	The service function for data link status control shall provide a mean to check if the PLC link will be terminated within TP_match_leave.

7.7.9 EIM status control functionality

In order to ensure EIM authorization, the MTC is able to detect and control EIM status information by using special functions depending on the MTC's type (SECC_Tester).

The parallel test component (PTC) for EIM status continuously observes the status of the EIM authorization until the authorization process is successful or failed.

Requirements of EIM status control for SECC testing

[V2G5-033]	The service function for EIM status control shall provide a mean to initiate the EIM authorization.
[V2G5-034]	The service function for EIM status control shall provide a mean to process incoming EIM status information from the EIM status PTC.

Requirements of EIM status PTC for SECC testing

[V2G5-035]	The EIM status PTC for SECC testing shall provide a function to inform the MTC if the EIM authorization is successful. If no change event occurs within a given time [ms], the function shall report an error.

7.7.10 Transmission power limitation functionality

In order to check the execution of amplitude map exchange on the SUTs node, the MTC shall be able to analyse the power level of the PLC signal.

Requirements of transmission power limitation for EVCC or SECC testing

[V2G5-036]	The service function for transmission power limitation shall provide a mean to check notched carriers in the frequency band 2 MHz to 30 MHz after a requested amplitude map exchange.

7.7.11 Attenuator injection functionality

In order to check that the received attenuation values within CM_ATTEN_CHAR.IND message are based on the physical channel, the MTC shall be able to reduce the level of signal on the control pilot.

Requirements of attenuator injection for SECC testing

[V2G5-037]	The service function for attenuator injection shall provide a mean to inject an RF attenuator on the control pilot, effective for the frequency band 2 MHz to 30 MHz.

8 Test case descriptions for ISO 15118-3 HPGP PLC signal measurement

8.1 General information

Subclause 8.2 covers the test procedure for PLC signal power spectrum density (PSD) measurement.

8.2 Test case for PLC signal measurement for ISO 15118-3

As defined in ISO 15118-3, the power level of the PLC signal of a SUT has to be calibrated according to the attenuation characteristics specific to the SUT. The corresponding calibration process and measurement procedure for HPGP is already defined in ISO 15118-3:2015, A.11.4. Hence, these PLC signal measurements are a prerequisite for the test cases described and specified in the ATS of this document in order to test conformance with ISO 15118-3.

[V2G5-038] Before the ATS specified in this standard is executed, the HPGP PLC signal measurement shall be performed according to ISO 15118-3:2015, A.11.4.2 for the SUT in order to test its conformance according to this document.

NOTE 1 Table 32 explicitly references all requirements that need to be considered within ISO 15118-3:2015, A.11.4.2.

Table 32 — Referenced requirements of ISO 15118-3 for HPGP PLC signal measurement procedure

Phase	Requirement
Setup phase (Preconditions for measurement)	[V2G3-A11-08] [V2G3-A11-09] [V2G3-A11-10] [V2G3-A11-11] [V2G3-A11-12] [V2G3-A11-14]
Measurement phase	[V2G3-A11-13] [V2G3-A11-15]

8.3 SECC + PLC bridge test cases

8.3.1 SECC test cases for CmSlacParm

8.3.1.1 Common test cases

Table 33 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_001'.

Table 33 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_001'

TC Id	`TC_SECC_CMN_VTB_CmSlacParm_001`
Test objective	Test System executes GoodCase procedure, indicates the initial CP State B transition independant of duty cycle. Test System then sends a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters and waits for the CM_SLAC_PARM.CNF message. Test System then checks that a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.3; 15118-3:A.9.3.1; 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:A.9.1.1; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-11], [V2G3-A09-15], [V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-A09-03], [V2G3-A09-01], [V2G3-A09-17]

Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_StateB_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacParm_001	

Table 34 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_002'.

Table 34 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_002'

TC Id	TC_SECC_CMN_VTB_CmSlacParm_002
Test objective	Test System executes GoodCase procedure, indicates the initial CP State C transition independant of duty cycle. Test System then sends a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters and waits for the CM_SLAC_PARM.CNF message. Test System then checks that a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.3; 15118-3:A.9.3.1; 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:A.9.1.1; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-11], [V2G3-A09-15], [V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-A09-03], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_StateB_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacParm_001	

Table 35 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_003'.

Table 35 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_003'

TC Id	TC_SECC_CMN_VTB_CmSlacParm_003
Test objective	Test System executes GoodCase procedure, indicates the initial CP State D transition independant of duty cycle. Test System then sends a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters and waits for the CM_SLAC_PARM.CNF message. Test System then checks that a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.3; 15118-3:A.9.3.1; 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:A.9.1.1; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-11], [V2G3-A09-15], [V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-A09-03], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001

PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_StateB_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacParm_001	

Table 36 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_004'.

Table 36 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_004'

TC Id	TC_SECC_CMN_VTB_CmSlacParm_004
Test objective	Test System executes GoodCase procedure and indicates the initial CP State B transition. Test System then waits until the TT_EVSE_SLAC_init timer (maximum value) has expired before sending the CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters. Test System then checks that no CM_SLAC_PARM.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.3.3; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.2.2
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-12], [V2G3-A09-13], [V2G3-A08-01], [V2G3-M06-11]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_StateB_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacParm_002	

Table 37 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_005'.

Table 37 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_005'

TC Id	TC_SECC_CMN_VTB_CmSlacParm_005
Test objective	Test System executes GoodCase procedure and indicates the initial CP State B transition. Test System then waits until the TT_EVSE_SLAC_init timer (maximum value) has expired before sending the CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters. Furthermore an additional CM_SLAC_PARM.REQ message with an invalid 'applicationType' equals to 'FF'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_PARM.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.3.3; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.2.2
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-12], [V2G3-A09-13], [V2G3-A09-14], [V2G3-A08-01], [V2G3-M06-11]
Config Id	CF_05_001
PICS selection	
PIXIT selection	

PreCondition
`f_SECC_CMN_PR_StateB_001`
Expected behavior
`f_SECC_CMN_TB_VTB_CmSlacParm_002`

Table 38 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_006'.

Table 38 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_006'

TC Id	`TC_SECC_CMN_VTB_CmSlacParm_006`
Test objective	Test System executes GoodCase procedure and indicates the initial CP State B transition. Test System then waits until the TT_EVSE_SLAC_init timer (maximum value) has expired before sending the CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters. Furthermore an additional CM_SLAC_PARM.REQ message with an invalid 'securityType' equals to 'FF'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_PARM.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.3.3; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.2.2
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-12], [V2G3-A09-13], [V2G3-A09-14], [V2G3-A08-01], [V2G3-M06-11]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_StateB_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacParm_002`	

Table 39 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_007'.

Table 39 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_007'

TC Id	`TC_SECC_CMN_VTB_CmSlacParm_007`
Test objective	Test System executes GoodCase procedure, indicates the initial CP State B transition, CP State A again and sends a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters. Test System then checks that no CM_SLAC_PARM.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8; 15118-3:A.9.1.1
Referenced requirement(s)	[V2G3-A09-126], [V2G3-A09-03]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_StateB_001`	

Expected behavior
f_SECC_CMN_TB_VTB_CmSlacParm_003

Table 40 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_008'.

Table 40— Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_008'

TC Id	TC_SECC_CMN_VTB_CmSlacParm_008
Test objective	Test System executes GoodCase procedure and stays for 'par_SECC_waitForPlugin' in CP State A (SUT goes to sleep after time period in CP State A). After 'par_SECC_waitForPlugin' Test System indicates the initial CP State B transition, sends CM_SLAC_PARM.REQ messages with a valid runID and all additional valid parameters and waits for the CM_SLAC_PARM.CNF message as long as 'par_T_conn_max_comm' is running. Test System then checks that a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters is sent by the SUT after wakeup by plug-in.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.3; 15118-3:A.9.3.1; 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:A.9.1.1; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-11], [V2G3-A09-15], [V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-A09-03], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_StateB_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacParm_004	

Table 41 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacParm_009'.

Table 41 — Test case description for 'TC_SECC_CMN_VTB_CmSlacParm_009'

TC Id	TC_SECC_CMN_VTB_CmSlacParm_009
Test objective	Test System executes GoodCase procedure and stays for 'par_SECC_waitForPlugin' in CP State A (SUT goes to sleep after time period in CP State A). After 'par_SECC_waitForPlugin' Test System indicates the initial CP State B transition and waits for a 5 % duty cycle as long as 'par_T_conn_max_comm' is running. After duty cycle detection Test System sends one CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters and waits for the CM_SLAC_PARM.CNF message. Test System then checks that a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters is sent by the SUT after wakeup by plug-in.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.3; 15118-3:A.9.3.1; 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:A.9.1.1; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-11], [V2G3-A09-15], [V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-M07-02], [V2G3-A09-03], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	

f_SECC_CMN_PR_StateB_001
Expected behavior
f_SECC_CMN_TB_VTB_CmSlacParm_005

8.3.1.2 AC specific test cases

Table 42 lists the test case description for 'TC_SECC_AC_VTB_CmSlacParm_001'.

Table 42 — Test case description for 'TC_SECC_AC_VTB_CmSlacParm_001'

TC Id	TC_SECC_AC_VTB_CmSlacParm_001
Test objective	Test System executes GoodCase procedure, indicates the initial CP State B, waits for 5 % duty cycle and executes successful EIM process. Test System then checks that a nominal duty cycle can be detected after SUT changes CP State to E or F for T_step_EF. Furthermore a successful SLAC matching process shall be performed afterwards.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:6.4.2.1; 15118-3:A.9; 15118-3:6.3; 15118-3:7.5.1.1
Referenced requirement(s)	[V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-M06-05], [V2G3-M06-09], [V2G3-A09-01], [V2G3-A09-17], [V2G3-M06-04], [V2G3-M06-08], [V2G3-M07-08]
Config Id	CF_05_001
PICS selection	PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_IdentificationMode := eIM, PICS_SECC_CMN_EIMDone := afterPlugin
PIXIT selection	PIXIT_SECC_AC_InitialDutyCyle := dc5
PreCondition	
f_SECC_CMN_PR_StateB_001	
Expected behavior	
f_SECC_AC_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_AttenuationCharacterization_001, f_SECC_CMN_TB_VTB_CmValidate_001, f_SECC_CMN_TB_VTB_CmSlacMatch_001, f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_CMN_TB_VTB_PLCLinkStatus_001	

Table 43 lists the test case description for 'TC_SECC_AC_VTB_CmSlacParm_002'.

Table 43 — Test case description for 'TC_SECC_AC_VTB_CmSlacParm_002'

TC Id	TC_SECC_AC_VTB_CmSlacParm_002
Test objective	Test System executes GoodCase procedure, indicates the initial CP State B transition and waits for 5 % duty cycle fallback if the TT_EVSE_SLAC_init timer has expired and no CM_SLAC_PARM.REQ messages was sent before. Test System then counts the number of fallback sequences. Test System then checks that a 5 % duty cycle can be detected after SUT changes CP State to E or F for T_step_EF and whether this sequence is limited to 2 retries by the SUT. When the repetition limit is reached, the Test System checks if the SUT is initiating a oscillator shutdown.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:6.4.2.1; 15118-3:A.9; 15118-3:6.3; 15118-3:7.5.1.1
Referenced requirement(s)	[V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-M06-07], [V2G3-M06-09], [V2G3-A09-01], [V2G3-A09-17], [V2G3-M06-04], [V2G3-M07-08]

Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_IdentificationMode := eIM`
PIXIT selection	`PIXIT_SECC_AC_InitialDutyCyle := dc5`
PreCondition	
`f_SECC_CMN_PR_StateB_001`	
Expected behavior	
`f_SECC_AC_TB_VTB_CmSlacParm_002`	

Table 44 lists the test case description for 'TC_SECC_AC_VTB_CmSlacParm_003'.

Table 44 — Test case description for 'TC_SECC_AC_VTB_CmSlacParm_003'

TC Id	`TC_SECC_AC_VTB_CmSlacParm_003`
Test objective	Test System executes GoodCase procedure, indicates the initial CP State B waits for 100 % duty cycle and executes successful EIM process. Test System then checks that a nominal duty cycle can be detected after SUT changes CP State to E or F for T_step_EF. Furthermore a successful SLAC matching process shall be performed afterwards.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:6.4.2.1; 15118-3:A.9; 15118-3:6.3; 15118-3:7.5.1.1
Referenced requirement(s)	[V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-M06-09], [V2G3-A09-01], [V2G3-A09-17], [V2G3-M06-04], [V2G3-M06-08], [V2G3-M07-08]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_IdentificationMode := eIM, PICS_SECC_CMN_EIMDone := afterPlugin`
PIXIT selection	`PIXIT_SECC_AC_InitialDutyCyle := dc100`
PreCondition	
`f_SECC_CMN_PR_StateB_001`	
Expected behavior	
`f_SECC_AC_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_AttenuationCharacterization_001, f_SECC_CMN_TB_VTB_CmValidate_001, f_SECC_CMN_TB_VTB_CmSlacMatch_001, f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_CMN_TB_VTB_PLCLinkStatus_001`	

Table 45 lists the test case description for 'TC_SECC_AC_VTB_CmSlacParm_004'.

Table 45 — Test case description for 'TC_SECC_AC_VTB_CmSlacParm_004'

TC Id	`TC_SECC_AC_VTB_CmSlacParm_004`
Test objective	Test System executes successful EIM process before plug-in, the GoodCase procedure and indicates the initial CP State B. Test System then checks that a nominal duty cycle can be detected. Furthermore a successful SLAC matching process shall be performed afterwards.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:A.9; 15118-3:6.3
Referenced requirement(s)	[V2G3-A09-18], [V2G3-M06-11], [V2G3-M07-01], [V2G3-A09-01], [V2G3-A09-17], [V2G3-M06-04]

Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_IdentificationMode := eIM, PICS_SECC_CMN_EIMDone := beforePlugin`
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_SetProximityPilot_001`	
Expected behavior	
`f_SECC_AC_TB_VTB_CmSlacParm_003, f_SECC_CMN_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_AttenuationCharacterization_001, f_SECC_CMN_TB_VTB_CmValidate_001, f_SECC_CMN_TB_VTB_CmSlacMatch_001, f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_CMN_TB_VTB_PLCLinkStatus_001`	

8.3.2 SECC test cases for AttenuationCharacterization

Table 46 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_001'.

Table 46 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_001'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_001`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then waits for the CM_ATTEN_CHAR.IND message. Test System then checks that a CM_ATTEN_CHAR.IND message with the current runID, EV MAC, 58 attenuation entries and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.3; 15118-3:A.9.2.1; 15118-3:A.9.2.2; 15118-3:A.9.3; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-18], [V2G3-M09-02], [V2G3-M09-04], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_001`	

Table 47 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_002'.

Table 47 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_002'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_002`
Test objective	Test System executes GoodCase procedure and sends a new CM_SLAC_PARM.REQ message with a new runID and all additional valid parameters after receiving a valid CM_ATTEN_CHAR.IND message with the current runID, 58 attenuation entries, EV MAC and all additional valid parameters. Test System then waits for the confirmation of a new SLAC process, receiving a CM_SLAC_PARM.CNF message because it shall be considered as a new retry by the EV.

	Test System then checks that a CM_SLAC_PARM.CNF message with current runID, EV MAC and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.1.3.3; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-16], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_002`	

Table 48 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_003'.

Table 48 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_003'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_003`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 9 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then waits for the CM_ATTEN_CHAR.IND message. Test System then checks that a CM_ATTEN_CHAR.IND message with the current runID, EV MAC, 58 attenuation entries and all additional valid parameters is sent by the SUT if the anticipated number of CM_ATTEN_PROFILE.IND messages is not achieved (n := 9 soundings).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.9.3; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-M09-02], [V2G3-M09-04], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_003`	

Table 49 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_004'.

Table 49 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_004'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_004`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND

	repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameter without sending a CM_ATTEN_CHAR.RSP message until the TT_match_response timer has expired. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_004`	

Table 50 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_005'.

Table 50 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_005'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_005`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameters after sending an invalid 'applicationType' equals to 'FF'H in CM_ATTEN_CHAR.RSP after each CM_ATTEN_CHAR.IND message. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A09-47], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 51 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_006'.

Table 51 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_006'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_006
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameters after sending an invalid 'securityType' equals to 'FF'H in CM_ATTEN_CHAR.RSP after each CM_ATTEN_CHAR.IND message. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A09-47], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_SECC_CMN_TB_VTB_AttenuationCharacterization_005	

Table 52 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_007'.

Table 52 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_007'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_007
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameters after sending an invalid 'sourceAddress' equals to '000000000000'H in CM_ATTEN_CHAR.RSP after each CM_ATTEN_CHAR.IND message. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A09-47], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	

`f_SECC_CMN_TB_VTB_AttenuationCharacterization_005`

Table 53 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_008'.

Table 53 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_008'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_008`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameters after sending an invalid 'runID' in CM_ATTEN_CHAR.RSP after each CM_ATTEN_CHAR.IND message. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A09-47], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_randomHexStringGen, f_SECC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 54 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_009'.

Table 54 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_009'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_009`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameters after sending an invalid 'sourceID' equals to '0000000000000000000000000000000001'H in CM_ATTEN_CHAR.RSP after each CM_ATTEN_CHAR.IND message. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A09-47], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001

PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 55 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_010'.

Table 55 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_010'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_010`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameters after sending an invalid 'respID' equals to '00000000000000000000000000000001'H in CM_ATTEN_CHAR.RSP after each CM_ATTEN_CHAR.IND message. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A09-47], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 56 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_011'.

Table 56 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_011'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_011`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then counts the number of CM_ATTEN_CHAR.IND repetitions including the current runID, EV MAC, 58 attenuation entries and all additional valid parameters after sending an invalid 'result' equals to 'FF'H in CM_ATTEN_CHAR.RSP after each CM_ATTEN_CHAR.IND message. Test System then checks the repetition of CM_ATTEN_CHAR.IND messages and whether it is limited to 2 retries by the SUT.
Document	Document: ISO:15118-3:2015:IS

reference	Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-42], [V2G3-A09-43], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-46], [V2G3-A09-47], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_SECC_CMN_TB_VTB_AttenuationCharacterization_005	

Table 57 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_012'.

Table 57 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_012'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_012
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters after the TT_match_sequence timer has expired. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_match_sequence timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-39], [V2G3-A09-40]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_SECC_CMN_TB_VTB_AttenuationCharacterization_006	

Table 58 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_013'.

Table 58 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_013'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_013
Test objective	Test System executes GoodCase procedure, sends an invalid 'applicationType' equals to 'FF'H in the 3 CM_START_ATTEN_CHAR.IND messages and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameter. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_EV_atten_results timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3
Referenced	[V2G3-A09-23], [V2G3-A09-41]

requirement(s)	
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_007`	

Table 59 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_014'.

Table 59 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_014'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_014`
Test objective	Test System executes GoodCase procedure, sends an invalid 'securityType' equals to 'FF'H in the 3 CM_START_ATTEN_CHAR.IND messages and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameter. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_EV_atten_results timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-41]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_007`	

Table 60 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_015'.

Table 60 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_015'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_015`
Test objective	Test System executes GoodCase procedure, sends an invalid 'numSounds' equals to '00'H in the 3 CM_START_ATTEN_CHAR.IND messages and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameter. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_EV_atten_results timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-41]
Config Id	CF_05_001
PICS selection	

PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_SECC_CMN_TB_VTB_AttenuationCharacterization_007	

Table 61 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_016'.

Table 61 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_016'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_016
Test objective	Test System executes GoodCase procedure, sends an invalid 'timeOut' equals to '00'H in the 3 CM_START_ATTEN_CHAR.IND messages and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameter. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_EV_atten_results timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-41]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_SECC_CMN_TB_VTB_AttenuationCharacterization_007	

Table 62 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_017'.

Table 62 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_017'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_017
Test objective	Test System executes GoodCase procedure, sends an invalid 'respType' equals to '00'H in the 3 CM_START_ATTEN_CHAR.IND messages and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameter. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_EV_atten_results timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-41]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	

Expected behavior
f_SECC_CMN_TB_VTB_AttenuationCharacterization_007

Table 63 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_018'.

Table 63 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_018'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_018
Test objective	Test System executes GoodCase procedure, sends an invalid 'runID' in the 3 CM_START_ATTEN_CHAR.IND messages and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_EV_atten_results timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.2; 15118-3:A.9.2.3.3
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-41]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_randomHexStringGen, f_SECC_CMN_TB_VTB_AttenuationCharacterization_007	

Table 64 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_019'.

Table 64 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_019'

TC Id	TC_SECC_CMN_VTB_AttenuationCharacterization_019
Test objective	Test System executes GoodCase procedure, indicates CP State A and sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters. Test System then checks that no CM_ATTEN_CHAR.IND message is sent by the SUT until the TT_EV_atten_results timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-126]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_SECC_CMN_TB_VTB_AttenuationCharacterization_008	

Table 65 lists the test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_020'.

Table 65 — Test case description for 'TC_SECC_CMN_VTB_AttenuationCharacterization_020'

TC Id	`TC_SECC_CMN_VTB_AttenuationCharacterization_020`
Test objective	Test System executes GoodCase procedure, sends 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters, so that the SUT can measure the individual attenuation values. Test System then waits for the CM_ATTEN_CHAR.IND message. Test System then checks that a CM_ATTEN_CHAR.IND message with the current runID, EV MAC, 58 attenuation entries and all additional valid parameters is sent by the SUT. Furthermore Test System executes a plug-out, injects an 10 dB attenuator on the control pilot and repeats the SLAC attenuation process by initiating a plug-in again. Test System then checks that SUT has sent the measured attenuation values by comparing both processes. (The mean of the attenuation values of test run 1 should be 'par_SECC_attenuationDeviation' smaller than the mean of the attenuation values of test run 2)
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.3; 15118-3:A.9.2.1; 15118-3:A.9.2.2; 15118-3:A.9.3; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-23], [V2G3-A09-44], [V2G3-A09-45], [V2G3-A09-18], [V2G3-M09-02], [V2G3-M09-04], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_001, f_SECC_CMN_Reset_001, f_SECC_CMN_TB_VTB_CmSlacParm_001, f_SECC_CMN_compareAttenuationValues_001`	

8.3.3 SECC test cases for CmValidate

Table 66 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_001'.

Table 66 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_001'

TC Id	`TC_SECC_CMN_VTB_CmValidate_001`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). Test System then sends a CM_VALIDATE.REQ message (step 2) with a valid 'pilotTimer' unequals to '00'H and all additional valid parameters, indicates the BCB toggle sequence and waits for the CM_VALIDATE.CNF message (step 2). Test System then checks that the CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters is sent by the SUT. Furthermore the Test System checks that the CM_VALIDATE.CNF message (step 2) with a valid number of toggles, 'result' equals to '02'H and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-79], [V2G3-A09-85], [V2G3-A09-86], [V2G3-A09-87], [V2G3-A08-01], [V2G3-M09-10], [V2G3-A09-56], [V2G3-A09-58], [V2G3-A09-59], [V2G3-A09-61], [V2G3-A09-01], [V2G3-A09-17]

Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_001`	

Table 67 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_002'.

Table 67 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_002'

TC Id	`TC_SECC_CMN_VTB_CmValidate_002`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). Test System then sends another CM_VALIDATE.REQ message (step 1) with a valid 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the repetition of CM_VALIDATE.CNF message (step 1). Test System then checks that a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters was repeated by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-79], [V2G3-A09-77], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_002`	

Table 68 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_003'.

Table 68 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_003'

TC Id	`TC_SECC_CMN_VTB_CmValidate_003`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). Test System then counts the number of CM_VALIDATE.CNF (step 1) repetitions including 'result' equals to '01'H and all additional valid parameters without sending a CM_VALIDATE.REQ message (step 2) message until the TT_match_sequence timer has expired. Test System then checks the repetition of CM_VALIDATE.CNF messages (step 1) and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-79], [V2G3-A09-82],

requirement(s)	[V2G3-A09-83], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_003`	

Table 69 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_004'.

Table 69 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_004'

TC Id	`TC_SECC_CMN_VTB_CmValidate_004`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). Test System then counts the number of CM_VALIDATE.CNF (step 1) repetitions including 'result' equals to '01'H and all additional valid parameters after sending an invalid 'signalType' equals to 'FF'H in CM_VALIDATE.REQ (step 2) after each CM_VALIDATE.CNF message (step 1). Test System then checks the repetition of CM_VALIDATE.CNF messages (step 1) and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-76], [V2G3-A09-79], [V2G3-A09-82], [V2G3-A09-83], [V2G3-A08-01], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_004`	

Table 70 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_005'.

Table 70 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_005'

TC Id	`TC_SECC_CMN_VTB_CmValidate_005`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). After receipt of a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters, Test System sends a CM_VALIDATE.REQ message (step 2) with result equals to '00'H and waits for termination of the SLAC matching process. Test System then checks that no CM_VALIDATE.CNF message is sent by the SUT until the TT_match_sequence timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-

	3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-76], [V2G3-A09-79], [V2G3-A08-01], [V2G3-A09-84], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_005`	

Table 71 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_006'.

Table 71 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_006'

TC Id	`TC_SECC_CMN_VTB_CmValidate_006`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). After receipt of a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters, Test System sends a CM_VALIDATE.REQ message (step 2) with result equals to '02'H and waits for termination of the SLAC matching process. Test System then checks that no CM_VALIDATE.CNF message is sent by the SUT until the TT_match_sequence timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-76], [V2G3-A09-79], [V2G3-A08-01], [V2G3-A09-84], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_005`	

Table 72 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_007'.

Table 72 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_007'

TC Id	`TC_SECC_CMN_VTB_CmValidate_007`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). After receipt of a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters, Test System sends a CM_VALIDATE.REQ message (step 2) with result equals to '03'H and waits for termination of the SLAC matching process. Test System then checks that no CM_VALIDATE.CNF message is sent by the SUT until the TT_match_sequence timer has expired.
Document	Document: ISO:15118-3:2015:IS

reference	Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-76], [V2G3-A09-79], [V2G3-A08-01], [V2G3-A09-84], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_005`	

Table 73 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_008'.

Table 73 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_008'

TC Id	`TC_SECC_CMN_VTB_CmValidate_008`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1). After receipt of a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters, Test System sends a CM_VALIDATE.REQ message (step 2) with result equals to '04'H and waits for termination of the SLAC matching process. Test System then checks that no CM_VALIDATE.CNF message is sent by the SUT until the TT_match_sequence timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-76], [V2G3-A09-79], [V2G3-A08-01], [V2G3-A09-84], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_005`	

Table 74 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_009'.

Table 74 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_009'

TC Id	`TC_SECC_CMN_VTB_CmValidate_009`
Test objective	Test System executes GoodCase procedure on a first instance and then executes a GoodCase procedure in parallel on a second instance while the first instance is waiting for CM_VALIDATE.REQ message (step 1). Then Test System executes the CM_VALIDATE.REQ message (step 1) sequence on the first instance, afterwards it sends the CM_VALIDATE.REQ message (step 1) on the second instance in parallel while the Test System is proceeding the validation process on the first instance. Test System then checks that the CM_VALIDATE.CNF message (step 1) with 'result'

	equals to '00'H and all additional valid parameters is sent to the second instance by the SUT (Processing blocked by first instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-79], [V2G3-A09-85], [V2G3-A09-86], [V2G3-A09-87], [V2G3-A08-01], [V2G3-M09-10], [V2G3-M09-13], [V2G3-A09-56], [V2G3-A09-58], [V2G3-A09-78], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001, f_SECC_CMN_TB_VTB_CmValidatePreCondition_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_006, f_SECC_CMN_TB_VTB_CmValidate_009, f_SECC_CMN_TB_VTB_CmValidate_007`	

Table 75 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_010'.

Table 75 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_010'

TC Id	`TC_SECC_CMN_VTB_CmValidate_010`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1) if the SUT is not able to perform any BCB-Toggle. Test System then checks that a CM_VALIDATE.CNF message with 'result' equals to '03'H and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A08-01], [V2G3-A09-53], [V2G3-A09-56], [V2G3-A09-58], [V2G3-A09-80], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_008`	

Table 76 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_011'.

Table 76 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_011'

TC Id	`TC_SECC_CMN_VTB_CmValidate_011`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1) if the SUT indicates that a validation is not

	required. Test System then checks that a CM_VALIDATE.CNF message (step 1) with 'result' equals to '04'H and all additional valid parameters is sent by the SUT. Furthermore Test System checks that the validation process can be finished if the EVCC continues the process by sending a CM_VALIDATE.REQ message (step 2) with a valid 'pilotTimer' unequals to '00'H and all additional valid parameters.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-79], [V2G3-A09-85], [V2G3-A09-86], [V2G3-A09-87], [V2G3-A08-01], [V2G3-M09-10], [V2G3-A09-56], [V2G3-A09-58], [V2G3-A09-59], [V2G3-A09-61], [V2G3-A09-81], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	

PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate,`
	`PIXIT_SECC_CMN_ArchitectureValidationNotRequired := true`

PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_008, f_SECC_CMN_TB_VTB_CmValidate_007`	

Table 77 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_012'.

Table 77 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_012'

TC Id	`TC_SECC_CMN_VTB_CmValidate_012`
Test objective	Test System executes GoodCase procedure, sends the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters and waits for the CM_VALIDATE.CNF message (step 1) if the SUT indicates that a validation is not required. Test System then checks that a CM_VALIDATE.CNF message (step 1) with 'result' equals to '04'H and all additional valid parameters is sent by the SUT. Furthermore Test System checks that the SUT responds to a valid CM_SLAC_MATCH.REQ message by sending a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC and all additional valid parameters if Test System skips the validation process.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.3; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-75], [V2G3-A09-79], [V2G3-A09-85], [V2G3-A09-86], [V2G3-A09-87], [V2G3-A08-01], [V2G3-M09-10], [V2G3-A09-56], [V2G3-A09-58], [V2G3-A09-59], [V2G3-A09-61], [V2G3-A09-81], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	

PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate,`
	`PIXIT_SECC_CMN_ArchitectureValidationNotRequired := true`

PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	

Expected behavior
`f_SECC_CMN_TB_VTB_CmValidate_008, f_SECC_CMN_TB_VTB_CmSlacMatch_001`

Table 78 lists the test case description for 'TC_SECC_CMN_VTB_CmValidate_013'.

Table 78 — Test case description for 'TC_SECC_CMN_VTB_CmValidate_013'

TC Id	`TC_SECC_CMN_VTB_CmValidate_013`
Test objective	Test System executes GoodCase procedure, indicates CP State A and sends a CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters. Test System then checks that no CM_VALIDATE.CNF message (step 1) is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-126]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmValidate_010`	

8.3.4 SECC test cases for CmSlacMatch

Table 79 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_001'.

Table 79 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_001'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_001`
Test objective	Test System executes GoodCase procedure without SLAC validation process, sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters and waits for CM_SLAC_MATCH.CNF message. Test System then checks that a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameter is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-99], [V2G3-A09-91], [V2G3-A09-52], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	

f_SECC_CMN_TB_VTB_CmSlacMatch_001

Table 80 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_002'.

Table 80 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_002'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_002
Test objective	Test System executes GoodCase procedure with SLAC validation process, sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parametersand waits for CM_SLAC_MATCH.CNF message. Test System then checks that a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameter is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-99], [V2G3-A09-91], [V2G3-A09-52], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := cmValidate
PreCondition	
f_SECC_CMN_PR_CmValidate_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacMatch_001	

Table 81 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_003'.

Table 81 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_003'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_003
Test objective	Test System executes GoodCase procedure without SLAC validation process, sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters and ignores the received CM_SLAC_MATCH.CNF message. After 'TT_match_response' timeout, Test System sends another CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters and waits for the repetition of a valid CM_SLAC_MATCH.CNF message. Test System then checks that a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameters is retransmitted by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-99], [V2G3-A09-91], [V2G3-A09-52], [V2G3-A09-97], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := none_
PreCondition	
f_SECC_CMN_PR_AttenuationCharacterization_001	

Expected behavior
f_SECC_CMN_TB_VTB_CmSlacMatch_002

Table 82 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_004'.

Table 82 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_004'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_004
Test objective	Test System executes GoodCase procedure with SLAC validation process, sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters and ignores the received CM_SLAC_MATCH.CNF message. After 'TT_match_response' timeout, Test System sends another CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters and waits for the repetition of a valid CM_SLAC_MATCH.CNF message. Test System then checks that a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameters is retransmitted by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-99], [V2G3-A09-91], [V2G3-A09-52], [V2G3-A09-97], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := cmValidate
PreCondition	
f_SECC_CMN_PR_CmValidate_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacMatch_002	

Table 83 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_005'.

Table 83 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_005'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_005
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := none_
PreCondition	
f_SECC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	

f_SECC_CMN_TB_VTB_CmSlacMatch_003

Table 84 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_006'.

Table 84 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_006'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_006
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.3.3.3
Referenced requirement(s)	[V2G3-A09-89], [V2G3-A09-90]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := cmValidate
PreCondition	
f_SECC_CMN_PR_CmValidate_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacMatch_003	

Table 85 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_007'.

Table 85 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_007'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_007
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'applicationType' equals to 'FF'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := none_
PreCondition	
f_SECC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacMatch_004	

Table 86 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_008'.

Table 86 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_008'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_008
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'applicationType' equals to 'FF'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := cmValidate
PreCondition	
f_SECC_CMN_PR_CmValidate_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacMatch_004	

Table 87 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_009'.

Table 87 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_009'

TC Id	TC_SECC_CMN_VTB_CmSlacMatch_009
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'securityType' equals to 'FF'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	PIXIT_SECC_CMN_CmValidate := none_
PreCondition	
f_SECC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmSlacMatch_004	

Table 88 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_010'.

Table 88 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_010'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_010`
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'securityType' equals to 'FF'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 89 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_011'.

Table 89 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_011'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_011`
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'mfvLength' equals to '0000'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 90 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_012'.

Table 90 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_012'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_012`

Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'mfvLength' equals to '0000'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 91 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_013'.

Table 91 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_013'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_013`
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evID' equals to '0000000000000000000000000000000001'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 92 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_014'.

Table 92 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_014'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_014`
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a

	CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evID' equals to '0000000000000000000000000000000001'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 93 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_015'.

Table 93 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_015'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_015`
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evMac' equals to '000000000000'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 94 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_016'.

Table 94 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_016'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_016`
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all

	additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evMac' equals to '000000000000'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 95 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_017'.

Table 95 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_017'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_017`
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evseID' equals to '00000000000000000000000000000001'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 96 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_018'.

Table 96 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_018'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_018`
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evseID'

	equals to '00000000000000000000000000000001'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 97 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_019'.

Table 97 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_019'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_019`
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evseMac' equals to '000000000000'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 98 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_020'.

Table 98 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_020'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_020`
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'evseMac'

	equals to '000000000000'H was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 99 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_021'.

Table 99 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_021'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_021`
Test objective	Test System executes GoodCase procedure without SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'runID' was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_randomHexStringGen, f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 100 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_022'.

Table 100 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_022'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_022`
Test objective	Test System executes GoodCase procedure with SLAC validation process and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters after TT_EVSE_match_session timer has expired. Furthermore an additional CM_SLAC_MATCH.REQ message with an invalid 'runID' was sent before timeout was triggered. Test System then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until

	the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.4.3.3
Referenced requirement(s)	[V2G3-A09-96], [V2G3-A09-98]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_randomHexStringGen, f_SECC_CMN_TB_VTB_CmSlacMatch_004`	

Table 101 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_023'.

Table 101 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_023'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_023`
Test objective	Test System executes GoodCase procedure without SLAC validation process, indicates CP State A and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters. Test system then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-126]
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := none_`
PreCondition	
`f_SECC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_005`	

Table 102 lists the test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_024'.

Table 102 — Test case description for 'TC_SECC_CMN_VTB_CmSlacMatch_024'

TC Id	`TC_SECC_CMN_VTB_CmSlacMatch_024`
Test objective	Test System executes GoodCase procedure with SLAC validation process, indicates CP State A and sends a CM_SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters. Test system then checks that no CM_SLAC_MATCH.CNF message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced	[V2G3-A09-126]

requirement(s)	
Config Id	CF_05_001
PICS selection	
PIXIT selection	`PIXIT_SECC_CMN_CmValidate := cmValidate`
PreCondition	
`f_SECC_CMN_PR_CmValidate_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmSlacMatch_005`	

8.3.5 SECC test cases for PLCLinkStatus

8.3.5.1 Common test cases

Table 103 lists the test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_001'.

Table 103 — Test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_001'

TC Id	`TC_SECC_CMN_VTB_PLCLinkStatus_001`
Test objective	Test System executes GoodCase procedure, indicates the key at the local PLC node after receipt of a valid CM_SLAC_MATCH.CNF message and then waits for AVLN establishment by exchange of communication with the internal PLC Node. Test System then checks that the data link connection is established within 'TT_match_join'.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:A.9.5.3.3
Referenced requirement(s)	[V2G3-A09-105]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_CmSetKey_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_001`	

Table 104 lists the test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_002'.

Table 104 — Test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_002'

TC Id	`TC_SECC_CMN_VTB_PLCLinkStatus_002`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Test System then checks that SUT leaves the logical network within 'TP_match_leave' if CP State A was detected before.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:9.7
Referenced requirement(s)	[V2G3-M09-19]
Config Id	CF_05_001
PICS selection	

PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_002`	

Table 105 lists the test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_003'.

Table 105 — Test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_003'

TC Id	`TC_SECC_CMN_VTB_PLCLinkStatus_003`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN and sends a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters. Test System then checks that no CM_SLAC_PARM.CNF message is sent by the SUT until the TT_match_response timer has expired if the SUT is in state 'Matched'.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-118], [V2G3-A09-03]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_003`	

Table 106 lists the test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_004'.

Table 106 — Test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_004'

TC Id	`TC_SECC_CMN_VTB_PLCLinkStatus_004`
Test objective	Test System starts GoodCase procedure five times in parallel and starts five independent SLAC processes. Test System then checks that the SUT will respond to each EVCC instance with a CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters. Futhermore a successful link establishment with the first instance will be checked.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.3; 15118-3:A.9.2.1; 15118-3:9.3; 15118-3:8; 15118-3:6.4.2.2; 15118-3:7.3.1; 15118-3:A.9.1.1; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-11], [V2G3-A09-15], [V2G3-A09-18], [V2G3-M09-01], [V2G3-M08-01], [V2G3-M06-11], [V2G3-M07-01], [V2G3-A09-03], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	

f_SECC_CMN_PR_StateB_001
Expected behavior
f_SECC_CMN_TB_VTB_AttenuationCharacterization_009, f_SECC_CMN_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_AttenuationCharacterization_001, f_SECC_CMN_TB_VTB_CmSlacMatch_001, f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_CMN_TB_VTB_PLCLinkStatus_001

Table 107 lists the test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_005'.

Table 107 — Test case description for 'TC_SECC_CMN_VTB_PLCLinkStatus_005'

TC Id	TC_SECC_CMN_VTB_PLCLinkStatus_005
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT leaves the logical network (setting previous key again on Test System side).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.7; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-34], [V2G3-M12-01]
Config Id	CF_05_001
PICS selection	
PIXIT selection	
PreCondition	
f_SECC_CMN_PR_PLCLinkStatus_001	
Expected behavior	
f_SECC_CMN_TB_VTB_PLCLinkStatus_009	

8.3.5.2 AC specific test cases

Table 108 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_001'.

Table 108 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_001'

TC Id	TC_SECC_AC_VTB_PLCLinkStatus_001
Test objective	Test System executes GoodCase procedure, initiates an EIM authorization during the SLAC process, indicates the key at the local PLC node and then checks that the data link connection is established within 'TT_match_join'. Test System then checks if the SUT has changed from 5 % initial duty cycle to a nominal duty cycle during the matching process by detecting X2(5 %) to X1(100 %) to X2(nominal) transition after successful EIM process during the matching process.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.5.3.3; 15118-3:6.4.2.1
Referenced requirement(s)	[V2G3-A09-105], [V2G3-M06-06]
Config Id	CF_05_001
PICS selection	PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_IdentificationMode := eIM, PICS_SECC_CMN_EIMDone := duringSlac
PIXIT selection	PIXIT_SECC_AC_InitialDutyCyle := dc5

PreCondition
`f_SECC_CMN_PR_PLCLinkStatus_001`
Expected behavior
`f_SECC_AC_TB_VTB_PLCLinkStatus_001`

Table 109 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_002'.

Table 109 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_002'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_002`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by initiating a BCB toggle and waits for successful data link detection triggered by the SUT. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (BCB toggle detection) and the successful data link detection within 'par_T_conn_resume'.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-20], [V2G3-M07-21], [V2G3-M07-22], [V2G3-M07-23], [V2G3-M07-26], [V2G3-M07-31], [V2G3-M12-01], [V2G3-M07-32]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	
PreCondition	
`f_SECC_AC_PR_SessionStop_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_004`	

Table 110 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_003'.

Table 110 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_003'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_003`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PIXIT_CMN_CMN_WakeUp' the SUT resumes the previously paused session by initiating a B1/B2 transition. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (initiating a B1/B2 transition) and the successful data link detection within 'par_T_conn_resume' and the processing of the SDP procedure (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-20], [V2G3-M07-24], [V2G3-M07-31], [V2G3-M09-16], [V2G3-M12-01], [V2G3-M07-32], [V2G3-M07-33]
Config Id	CF_05_001

PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	
PreCondition	
`f_SECC_AC_PR_SessionStop_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_005`	

Table 111 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_004'.

Table 111 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_004'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_004`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by initiating a BCB toggle and waits for failed link detection (new logical network parameter was set from the Test System before). Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks that the SUT applys CP State E or F for T_step_EF and will process the start of a new matching process.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:7.5.1.1
Referenced requirement(s)	[V2G3-M07-27], [V2G3-M07-31], [V2G3-M07-08], [V2G3-M07-32]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	
PreCondition	
`f_SECC_AC_PR_SessionStop_001, f_SECC_startSleepingPhase, f_randomHexStringGen, f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_006`	

Table 112 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_005'.

Table 112 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_005'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_005`
Test objective	Test System executes GoodCase procedure (5 % duty cycle), indicates the key at the local PLC node and then checks that the data link connection is established within 'TT_match_join'. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT performs a CP State X2 to X1 to E/F transition. Furthermore Test System checks that the SUT applies CP State E or F for T_step_EF and will switch to CP State X1 or X2 afterwards.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.1.1

Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-05], [V2G3-M07-06], [V2G3-M07-07], [V2G3-M07-08], [V2G3-M07-09]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC`
PIXIT selection	`PIXIT_SECC_AC_InitialDutyCyle := dc5`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_007`	

Table 113 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_006'.

Table 113 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_006'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_006`
Test objective	Test System executes GoodCase procedure (Nominal duty cycle), indicates the key at the local PLC node and then checks that the data link connection is established within 'TT_match_join'. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT performs a CP State X2 to X1 to E/F transition. Furthermore Test System checks that the SUT applys CP State E or F for T_step_EF and will switch to CP State X1 or X2 afterwards (Option A).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.1.1; 15118-3:7.7; 15118-3:7.5.1.2
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-05], [V2G3-M07-06], [V2G3-M07-07], [V2G3-M07-08], [V2G3-M07-09], [V2G3-M07-34], [V2G3-M07-10], [V2G3-M07-11]
Config Id	CF_05_001

PICS selection	`PICS_SECC_CMN_EIMDone := afterPlugin,`
	`PICS_CMN_CMN_ChargingMode := aC,`
	`PICS_CMN_CMN_IdentificationMode := eIM`

PIXIT selection	`PIXIT_SECC_AC_InitialDutyCyle := dc5,`
	`PIXIT_SECC_AC_ConnectionLossHandling := optionA`

PreCondition	
`f_SECC_CMN_PR_StateB_001`	
Expected behavior	

`f_SECC_AC_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_CmSlacParm_001,`
`f_SECC_CMN_TB_VTB_AttenuationCharacterization_001,`
`f_SECC_CMN_TB_VTB_CmValidate_001, f_SECC_CMN_TB_VTB_CmSlacMatch_001,`
`f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_CMN_TB_VTB_PLCLinkStatus_001,`
`f_SECC_CMN_TB_VTB_PLCLinkStatus_007`	

Table 114 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_007'.

Table 114 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_007'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_007`
Test objective	Test System executes GoodCase procedure (Nominal duty cycle), indicates the key at the local PLC node and then checks that the data link connection is established within 'TT_match_join'. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT stays in CP State X2 and will process the start of a

	new matching process (Option B).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.7; 15118-3:7.5.1.2
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-34], [V2G3-M07-10], [V2G3-M07-12]
Config Id	CF_05_001

PICS selection	`PICS_SECC_CMN_EIMDone := afterPlugin,`
	`PICS_CMN_CMN_ChargingMode := aC,`
	`PICS_CMN_CMN_IdentificationMode := eIM`

PIXIT selection	`PIXIT_SECC_AC_InitialDutyCyle := dc5,`
	`PIXIT_SECC_AC_ConnectionLossHandling := optionB`

PreCondition	
`f_SECC_CMN_PR_StateB_001`	
Expected behavior	
`f_SECC_AC_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_CmSlacParm_001, f_SECC_CMN_TB_VTB_AttenuationCharacterization_001, f_SECC_CMN_TB_VTB_CmValidate_001, f_SECC_CMN_TB_VTB_CmSlacMatch_001, f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_CMN_TB_VTB_PLCLinkStatus_001, f_SECC_AC_TB_VTB_PLCLinkStatus_002`	

Table 115 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_008'.

Table 115 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_008'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_008`
Test objective	Test System executes GoodCase procedure and initiates a terminated V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Terminate' and all additional mandatory parameters and waits for a SessionStopRes message with the current SessionID and all additional mandatory parameters. Furthermore Test System checks if the SUT terminates the TCP connection after. Test System then checks that the SUT leaves the logical network after 'TP_match_leave' by using the CM_NW_STATS message sequence.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.7; 15118-3:A.9.7
Referenced requirement(s)	[V2G3-M09-17], [V2G3-A09-121]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_CombinedTesting := true, PICS_SECC_CMN_Pause := false`
PIXIT selection	
PreCondition	
`f_SECC_AC_PR_SessionStop_002`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_008`	

Table 116 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_009'.

Table 116 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_009'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_009`

Test objective	Test System executes GoodCase procedure, initiates an EIM authorization during the SLAC process, indicates the key at the local PLC node and then checks that the data link connection is established within 'TT_match_join'. Test System then checks if the SUT has changed from 100 % initial duty cycle to a nominal duty cycle during the matching process by detecting X1(100 %) to X2(nominal) transition after successful EIM process during the matching process.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.5.3.3
Referenced requirement(s)	[V2G3-A09-105]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_IdentificationMode := eIM, PICS_SECC_CMN_EIMDone := duringSlac`
PIXIT selection	`PIXIT_SECC_AC_InitialDutyCyle := dc100`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_AC_TB_VTB_PLCLinkStatus_001`	

Table 117 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_010'.

Table 117 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_010'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_010`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (BCB toggle detection), the B1/B2 transition by the oscillator and the processing of the SDP procedure (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-20], [V2G3-M07-25], [V2G3-M07-31], [V2G3-M09-16], [V2G3-M12-01], [V2G3-M07-32]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	
PreCondition	
`f_SECC_AC_PR_SessionStop_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_010`	

Table 118 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_011'.

Table 118 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_011'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_011`
Test objective	Test System executes GoodCase procedure (100 % duty cycle), indicates the key at the local PLC node and then checks that the data link connection is established within 'TT_match_join'. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT performs a CP State X1 to E/F transition. Furthermore Test System checks that the SUT applies CP State E or F for T_step_EF and will switch to CP State X1 or X2 afterwards.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.1.1
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-04], [V2G3-M07-08], [V2G3-M07-09]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC`
PIXIT selection	`PIXIT_SECC_AC_InitialDutyCyle := dc100`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_011`	

Table 119 lists the test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_012'.

Table 119 — Test case description for 'TC_SECC_AC_VTB_PLCLinkStatus_012'

TC Id	`TC_SECC_AC_VTB_PLCLinkStatus_012`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' < 'par_SECC_T_step_X1' Test System resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (BCB toggle detection) and the B1/B2 transition by the oscillator. As a result of 'PICS_CMN_CMN_WakeUp' < 'par_SECC_T_step_X1', SUT shall signal B1/B2 transition not earlier than 'par_SECC_T_step_X1' within B1.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-20], [V2G3-M07-25], [V2G3-M07-31], [V2G3-M09-16], [V2G3-M12-01], [V2G3-M07-32], [V2G3-M07-33]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true, PICS_CMN_CMN_WakeUp < par_SECC_T_step_X1`
PIXIT selection	
PreCondition	
`f_SECC_AC_PR_SessionStop_001, f_SECC_startSleepingPhase`	
Expected behavior	

f_SECC_CMN_TB_VTB_PLCLinkStatus_012

8.3.5.3 DC specific test cases

Table 120 lists the test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_001'.

Table 120 — Test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_001'

TC Id	TC_SECC_DC_VTB_PLCLinkStatus_001
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by initiating a BCB toggle and waits for successful data link detection triggered by the SUT. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (BCB toggle detection) and the successful data link detection within 'par_T_conn_resume'.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-20], [V2G3-M07-21], [V2G3-M07-22], [V2G3-M07-23], [V2G3-M07-26], [V2G3-M07-31], [V2G3-M12-01], [V2G3-M07-32]
Config Id	CF_05_001
PICS selection	PICS_CMN_CMN_ChargingMode := dC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true
PIXIT selection	
PreCondition	
f_SECC_DC_PR_SessionStop_001, f_SECC_startSleepingPhase	
Expected behavior	
f_SECC_CMN_TB_VTB_PLCLinkStatus_004	

Table 121 lists the test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_002'.

Table 121 — Test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_002'

TC Id	TC_SECC_DC_VTB_PLCLinkStatus_002
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PIXIT_CMN_CMN_WakeUp' the SUT resumes the previously paused session by initiating 5 % duty cycle. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (initiating 5 % duty cycle) and the successful data link detection within 'par_T_conn_resume' and the processing of the SDP procedure (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-20], [V2G3-M07-24], [V2G3-M07-31], [V2G3-M09-16], [V2G3-M12-01], [V2G3-M07-32], [V2G3-M07-33]
Config Id	CF_05_001
PICS selection	PICS_CMN_CMN_ChargingMode := dC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp,

	`PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	
PreCondition	
`f_SECC_DC_PR_SessionStop_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_005`	

Table 122 lists the test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_003'.

Table 122 — Test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_003'

TC Id	`TC_SECC_DC_VTB_PLCLinkStatus_003`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by initiating a BCB toggle and waits for failed link detection (new logical network parameter was set from the Test System before). Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks that the SUT applys CP State E or F for T_step_EF and will process the start of a new matching process.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:7.5.1.1
Referenced requirement(s)	[V2G3-M07-27], [V2G3-M07-31], [V2G3-M07-08], [V2G3-M07-32]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := dC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	
PreCondition	
`f_SECC_DC_PR_SessionStop_001, f_SECC_startSleepingPhase, f_randomHexStringGen, f_SECC_CMN_TB_VTB_CmSetKey_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_006`	

Table 123 lists the test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_004'.

Table 123 — Test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_004'

TC Id	`TC_SECC_DC_VTB_PLCLinkStatus_004`
Test objective	Test System executes GoodCase procedure, indicates the key at the local PLC node and then checks that the data link connection is established within 'TT_match_join'. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT performs a CP State X2 to X1 to E/F transition. Furthermore Test System checks that the SUT applys CP State E or F for T_step_EF and will switch to CP State X1 or X2 afterwards.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.1.1
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-05], [V2G3-M07-06], [V2G3-M07-07], [V2G3-M07-08], [V2G3-M07-09]
Config Id	CF_05_001

PICS selection	`PICS_CMN_CMN_ChargingMode := dC`
PIXIT selection	
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_007`	

Table 124 lists the test case **description** for 'TC_SECC_DC_VTB_PLCLinkStatus_005'.

Table 124 — Test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_005'

TC Id	`TC_SECC_DC_VTB_PLCLinkStatus_005`
Test objective	Test System executes GoodCase procedure and initiates a terminated V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Terminate' and all additional mandatory parameters and waits for a SessionStopRes message with the current SessionID and all additional mandatory parameters. Furthermore Test System checks if the SUT terminates the TCP connection after successful charging process. Test System then checks that the SUT leaves the logical network after 'TP_match_leave' by using the CM_NW_STATS message sequence.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.7; 15118-3:A.9.7
Referenced requirement(s)	[V2G3-M09-17], [V2G3-A09-121]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := dC, PICS_CMN_CMN_CombinedTesting := true, PICS_SECC_CMN_Pause := false`
PIXIT selection	
PreCondition	
`f_SECC_DC_PR_SessionStop_002`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_008`	

Table 125 lists the test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_006'.

Table 125 — Test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_006'

TC Id	`TC_SECC_DC_VTB_PLCLinkStatus_006`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (BCB toggle detection), the 5 % duty cycle detection and the processing of the SDP procedure (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:9.5; 15118-3:12.3
Referenced	[V2G3-M07-20], [V2G3-M07-25], [V2G3-M07-31], [V2G3-M09-16], [V2G3-M12-01],

requirement(s)	[V2G3-M07-32]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := dC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	
PreCondition	
`f_SECC_DC_PR_SessionStop_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_010`	

Table 126 lists the test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_007'.

Table 126 — Test case description for 'TC_SECC_DC_VTB_PLCLinkStatus_007'

TC Id	`TC_SECC_DC_VTB_PLCLinkStatus_007`
Test objective	Test System executes GoodCase procedure and initiates a paused V2G communication session by sending a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' < 'par_SECC_T_step_X1' Test System resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT shall not turn off the +12 V supply during the sleeping phase. Furthermore Test System checks the wake-up process (BCB toggle detection) and the 5 % duty cycle detection by the oscillator. As a result of 'PICS_CMN_CMN_WakeUp' < 'par_SECC_T_step_X1', SUT shall signal 5 % duty cycle not earlier than 'par_SECC_T_step_X1' within B1.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2.1; 15118-3:7.6.3; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-20], [V2G3-M07-25], [V2G3-M07-31], [V2G3-M09-16], [V2G3-M12-01], [V2G3-M07-32], [V2G3-M07-33]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_ChargingMode := dC, PICS_SECC_CMN_Pause := true, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true, PICS_CMN_CMN_WakeUp < 'par_SECC_T_step_X1'`
PIXIT selection	
PreCondition	
`f_SECC_DC_PR_SessionStop_001, f_SECC_startSleepingPhase`	
Expected behavior	
`f_SECC_CMN_TB_VTB_PLCLinkStatus_012`	

8.3.6 SECC test cases for CmAmpMap

Table 127 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_001'.

Table 127 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_001'

TC Id	`TC_SECC_CMN_VTB_CmAmpMap_001`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN and sends CM_AMP_MAP.REQ message with a new amplitude map and all additional valid parameters. Test System then checks that a CM_AMP_MAP.CNF message with 'result' equals to '00'H

	is sent by the SUT. Furthermore the reduction of the transmission power of the requested carriers will be checked with additional equipment.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmAmpMap_001, f_SECC_CMN_checkTXPowerLimitation`	

Table 128 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_002'.

Table 128 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_002'

TC Id	`TC_SECC_CMN_VTB_CmAmpMap_002`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Test System then checks that CM_AMP_MAP.REQ message with a new amplitude map and all additional valid parameters is sent by the SUT until 'par_TT_amp_map_exchange' has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-109], [V2G3-A09-111], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := false`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmAmpMap_002`	

Table 129 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_003'.

Table 129 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_003'

TC Id	`TC_SECC_CMN_VTB_CmAmpMap_003`
Test objective	Test System executes GoodCase procedure and counts the number of CM_AMP_MAP.REQ repetitions including a new amplitude map and all additional valid parameter without sending a CM_AMP_MAP.CNF message until the TT_match_response timer has expired. Test System then checks the repetition of CM_AMP_MAP.REQ messages and whether it is limited to 2 retries by the SUT.
Document	Document: ISO:15118-3:2015:IS

reference	Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-109], [V2G3-A09-111], [V2G3-A09-112]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := false`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmAmpMap_003`	

Table 130 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_004'.

Table 130 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_004'

TC Id	`TC_SECC_CMN_VTB_CmAmpMap_004`
Test objective	Test System executes GoodCase procedure and counts the number of CM_AMP_MAP.REQ repetitions including a new amplitude map and all additional valid parameter after sending an invalid 'result' equals to 'FF'H in the CM_AMP_MAP.CNF messages. Test System then checks the repetition of CM_AMP_MAP.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-109], [V2G3-A09-111], [V2G3-A09-112], [V2G3-A09-114]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := false`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmAmpMap_004`	

Table 131 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_005'.

Table 131 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_005'

TC Id	`TC_SECC_CMN_VTB_CmAmpMap_005`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN and sends an invalid CM_AMP_MAP.REQ message with 'amLen' equals to '00'H and all additional valid parameters. Test System then checks that no CM_AMP_MAP.CNF message is sent by the SUT until TT_match_response timer has expired. This sequence will be repeated for 2 retries.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-115], [V2G3-A09-113]
Config Id	CF_05_001

PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmAmpMap_005`	

Table 132 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_006'.

Table 132 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_006'

TC Id	`TC_SECC_CMN_VTB_CmAmpMap_006`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN, sends a CM_AMP_MAP.REQ message with a new amplitude map and all additional valid parameters and waits for a valid CM_AMP_MAP.CNF message. Test System then sends another valid CM_AMP_MAP.REQ message and checks that a CM_AMP_MAP.CNF message with 'result' equals to '00'H and all additional valid parameters is sent by the SUT again.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-116]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_SECC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_SECC_CMN_TB_VTB_CmAmpMap_006`	

Table 133 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_007'.

Table 133 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_007'

TC Id	`TC_SECC_CMN_VTB_CmAmpMap_007`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN and sends a burst of 3 CM_AMP_MAP.REQ messages with a new amplitude map and all additional valid parameters. Test System then checks that a CM_AMP_MAP.CNF message with 'result' equals to '00'H and all additional valid parameters is sent by the SUT for each request message.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-116]
Config Id	CF_05_001
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	

f_SECC_CMN_PR_PLCLinkStatus_001
Expected behavior
f_SECC_CMN_TB_VTB_CmAmpMap_007

Table 134 lists the test case description for 'TC_SECC_CMN_VTB_CmAmpMap_008'.

Table 134 — Test case description for 'TC_SECC_CMN_VTB_CmAmpMap_008'

TC Id	TC_SECC_CMN_VTB_CmAmpMap_008
Test objective	Test System executes GoodCase procedure, initiates an amplitude map process with transmission power limitation check and sends a valid SDP request message. Test System then checks that the SUT sends a valid SDP response message. Furthermore the usability of the matched bandwidth (number of allocatable carriers) will be checked after amplitude map process.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_001
PICS selection	PICS_CMN_CMN_InitiateCmAmpMap := true
PIXIT selection	PIXIT_CMN_CMN_CmAmpMap := true
PreCondition	
f_SECC_CMN_PR_CmAmpMap_001	
Expected behavior	
f_SECC_CMN_TB_VTB_CmAmpMap_008	

8.4 EVCC + PLC bridge test cases

8.4.1 EVCC test cases for CmSlacParm

8.4.1.1 Common test cases

Table 135 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_001'.

Table 135 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_001'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_001
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' (transition E to X2) after initial CP State B transition and waits for CM_SLAC_PARM.REQ. Test System then checks that a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	

PreCondition
f_EVCC_CMN_PR_DutyCycle_001
Expected behavior
f_EVCC_CMN_TB_VTB_CmSlacParm_001

Table 136 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_002'.

Table 136 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_002'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_002
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters without sending a CM_SLAC_PARM.CNF message until the TT_match_response timer has expired. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_002	

Table 137 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_003'.

Table 137 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_003'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_003
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters after sending an invalid 'mSoundTarget' equals to '000000000000'H in CM_SLAC_PARM.CNF after each CM_SLAC_PARM.REQ message. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-09], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	

PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacParm_003`	

Table 138 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_004'.

Table 138 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_004'

TC Id	`TC_EVCC_CMN_VTB_CmSlacParm_004`
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters after sending an invalid 'timeout' equals to '00'H in CM_SLAC_PARM.CNF after each CM_SLAC_PARM.REQ message. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-09], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacParm_003`	

Table 139 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_005'.

Table 139 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_005'

TC Id	`TC_EVCC_CMN_VTB_CmSlacParm_005`
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters after sending an invalid 'respType' equals to '00'H in CM_SLAC_PARM.CNF after each CM_SLAC_PARM.REQ message. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-09], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002

PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_003	

Table 140 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_006'.

Table 140 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_006'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_006
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters after sending an invalid 'applicationType' equals to 'FF'H in CM_SLAC_PARM.CNF after each CM_SLAC_PARM.REQ message. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-09], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_003	

Table 141 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_007'.

Table 141 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_007'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_007
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters after sending an invalid 'securityType' equals to 'FF'H in CM_SLAC_PARM.CNF after each CM_SLAC_PARM.REQ message. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-09], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01],

	[V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_003	

Table 142 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_008'.

Table 142 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_008'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_008
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters after sending an invalid 'runID' in CM_SLAC_PARM.CNF after each CM_SLAC_PARM.REQ message. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-09], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_randomHexStringGen, f_EVCC_CMN_TB_VTB_CmSlacParm_003	

Table 143 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_009'.

Table 143 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_009'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_009
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay'after initial CP State B transition and waits for a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters. After receiving a valid CM_SLAC_PARM.REQ message, Test System then checks that the SUT will not respond to the following messages: CM_SLAC_PARM.REQ, CM_START_ATTEN_CHAR.IND, CM_MNBC_SOUND.IND, CM_ATTEN_CHAR.RSP, CM_VALIDATE.REQ, CM_SLAC_MATCH.REQ.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.6.5.2; 15118-3:A.9
Referenced	[V2G3-A09-04], [V2G3-A06-05], [V2G3-A09-01], [V2G3-A09-17]

requirement(s)	
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacParm_004`	

Table 144 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_010'.

Table 144 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_010'

TC Id	`TC_EVCC_CMN_VTB_CmSlacParm_010`
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters without sending a CM_SLAC_PARM.CNF message until the TT_match_response timer has expired. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT and if this sequence is retried as long as the TT_Matching_Repetition timer is running.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9.8; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-122], [V2G3-A09-123], [V2G3-A09-124], [V2G3-A09-125], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	

PIXIT selection	`PIXIT_EVCC_CMN_TTMatchingRepetitionConfig := true,`
	`PIXIT_EVCC_CMN_TTMatchingRepetition,`
	`PIXIT_EVCC_CMN_TTMatchingRate`

PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacParm_005`	

Table 145 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_011'.

Table 145 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_011'

TC Id	`TC_EVCC_CMN_VTB_CmSlacParm_011`
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay'after initial CP State B transition and counts the number of CM_SLAC_PARM.REQ repetitions including a valid runID and all additional valid parameters after sending an invalid 'forwardingSta' equals to '000000000000'H in CM_SLAC_PARM.CNF after each CM_SLAC_PARM.REQ message. Test System then checks the repetition of CM_SLAC_PARM.REQ messages and whether it is limited to 2 retries by the SUT.

Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:A.8; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-A08-01], [V2G3-A09-07], [V2G3-A09-08], [V2G3-A09-09], [V2G3-A09-10], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_003	

Table 146 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_012'.

Table 146 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_012'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_012
Test objective	Test System executes GoodCase procedure, indicates a 5 % duty cycle with a delay of 'par_EVCC_setDC_delay' (transition F to X2) after initial CP State B transition and waits for CM_SLAC_PARM.REQ. Test System then checks that a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_001	

Table 147 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_013'.

Table 147 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_013'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_013
Test objective	Test System executes GoodCase procedure, indicates a 100 % duty cycle with a delay of 'par_EVCC_setDC_delay' (transition E to X1) after initial CP State B transition and waits for CM_SLAC_PARM.REQ. Test System then checks that a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:6.4.3.2; 15118-3:A.9

Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_001	

Table 148 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_014'.

Table 148 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacParm_014'

TC Id	TC_EVCC_CMN_VTB_CmSlacParm_014
Test objective	Test System executes GoodCase procedure, indicates a 100 % duty cycle with a delay of 'par_EVCC_setDC_delay' (transition F to X1) after initial CP State B transition and waits for CM_SLAC_PARM.REQ. Test System then checks that a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_001	

8.4.1.2 AC specific test cases

Table 149 lists the test case description for 'TC_EVCC_AC_VTB_CmSlacParm_001'.

Table 149 — Test case description for 'TC_EVCC_AC_VTB_CmSlacParm_001'

TC Id	TC_EVCC_AC_VTB_CmSlacParm_001
Test objective	Test System executes GoodCase procedure, indicates a 53 % duty cycle (32A) with a delay of 'par_EVCC_setDC_delay' (transition E to X2) after initial CP State B transition and waits for CM_SLAC_PARM.REQ. Test System then checks that a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:6.4.3.2; 15118-3:A.9
Referenced	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-M06-13], [V2G3-A09-01],

requirement(s)	[V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_001	

Table 150 lists the test case description for 'TC_EVCC_AC_VTB_CmSlacParm_002'.

Table 150 — Test case description for 'TC_EVCC_AC_VTB_CmSlacParm_002'

TC Id	TC_EVCC_AC_VTB_CmSlacParm_002
Test objective	Test System executes GoodCase procedure, indicates a 53 % duty cycle (32A) with a delay of 'par_EVCC_setDC_delay' (transition F to X2) after initial CP State B transition and waits for CM_SLAC_PARM.REQ. Test System then checks that a CM_SLAC_PARM.REQ message with a valid runID and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.1.2; 15118-3:A.9.1.3.2; 15118-3:A.9.2.1; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-04], [V2G3-A09-05], [V2G3-A09-18], [V2G3-M06-13], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacParm_001	

8.4.2 EVCC test cases for AttenuationCharacterization

8.4.2.1 Common test cases

Table 151 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_001'.

Table 151 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_001'

TC Id	TC_EVCC_CMN_VTB_AttenuationCharacterization_001
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT. Test System then checks that 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters are sent by the SUT. Furthermore the SUT shall confirm the attenuation values by sending the CM_ATTEN_CHAR.RSP message with the current runID, EV MAC

	and all additional valid parameters.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9.3; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-37], [V2G3-M09-02], [V2G3-M09-03], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001`	

Table 152 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_002'.

Table 152 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_002'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_002`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT. This procedure is started twice in parallel but in the second instance the CM_SLAC_PARM.CNF message will not be send. Test System then checks that a CM_ATTEN_CHAR.RSP message with the current runID, EV MAC and all additional valid parameters is sent by the SUT if a CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters is received and its origin is an EVSE that has not sent a CM_SLAC_PARM.CNF message before.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9.3; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-37], [V2G3-A09-33], [V2G3-M09-02], [V2G3-M09-03], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	

`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_002,`
`f_EVCC_CMN_TB_VTB_CmSlacParm_001,`
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001`	

Table 153 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_003'.

Table 153 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_003'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_003`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message

	with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_004`	

Table 154 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_004'.

Table 154 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_004'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_004`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'applicationType' equals to 'FF'H was sent before the timer expires. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 155 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_005'.

Table 155 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_005'

TC Id	TC_EVCC_CMN_VTB_AttenuationCharacterization_005
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'securityType' equals to 'FF'H was sent before the timer expires. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005	

Table 156 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_006'.

Table 156 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_006'

TC Id	TC_EVCC_CMN_VTB_AttenuationCharacterization_006
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'sourceAddress' equals to '000000000000'H was sent before the timer expires. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_CmSlacParm_001	

Expected behavior
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005`

Table 157 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_007'.

Table 157 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_007'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_007`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'runID' was sent before timeout was triggered. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_randomHexStringGen, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 158 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_008'.

Table 158 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_008'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_008`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'sourceID' equals to '00000000000000000000000000000001'H was sent before timeout was triggered. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002

PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 159 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_009'.

Table 159 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_009'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_009`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'respID' equals to '00000000000000000000000000000001'H was sent before timeout was triggered. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 160 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_010'.

Table 160 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_010'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_010`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'numGroups' equals to '00'H was sent before timeout was triggered. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.

Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 161 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_011'.

Table 161 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_011'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_011`
Test objective	Test System executes GoodCase procedure, sends the CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters and waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages, so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT after the TT_EV_atten_results timer has expired. Furthermore an additional CM_ATTEN_CHAR.IND message with an invalid 'numSounds' equals to '00'H was sent before timeout was triggered. Test System then checks that no CM_ATTEN_CHAR.RSP message is sent by the SUT until the TT_match_response timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-30], [V2G3-A09-31], [V2G3-A09-32], [V2G3-A09-35], [V2G3-A09-36], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005`	

Table 162 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_012'.

Table 162 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_012'

TC Id	`TC_EVCC_CMN_VTB_AttenuationCharacterization_012`
Test objective	Test System executes GoodCase procedure, signals CP State E and sends a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters. Test System then checks that no CM_START_ATTEN_CHAR.IND message is sent by the SUT until the TT_match_sequence timer has expired.

Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-127]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_AttenuationCharacterization_006	

Table 163 lists the test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_013'.

Table 163 — Test case description for 'TC_EVCC_CMN_VTB_AttenuationCharacterization_013'

TC Id	TC_EVCC_CMN_VTB_AttenuationCharacterization_013
Test objective	Test System executes GoodCase procedure, signals CP State F and sends a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters. Test System then checks that no CM_START_ATTEN_CHAR.IND message is sent by the SUT until the TT_match_sequence timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-127]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_CmSlacParm_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_AttenuationCharacterization_006	

8.4.2.2 AC specific test cases

Table 164 lists the test case description for 'TC_EVCC_AC_VTB_AttenuationCharacterization_001'.

Table 164 — Test case description for 'TC_EVCC_AC_VTB_AttenuationCharacterization_001'

TC Id	TC_EVCC_AC_VTB_AttenuationCharacterization_001
Test objective	Test System executes GoodCase procedure, changes the duty cycle to 10 % before sending a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters. Test System then waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT. Test System then checks that 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND

	messages with the current runID, a decrementing counter and all additional valid parameters are sent by the SUT. Furthermore the SUT shall confirm the attenuation values by sending the CM_ATTEN_CHAR.RSP message with the current runID, EV MAC and all additional valid parameters.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-37], [V2G3-M06-15], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002

PICS selection	`PICS_CMN_CMN_ChargingMode := aC,`
	`PICS_CMN_CMN_IdentificationMode := eIM`

PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_AC_TB_VTB_AttenuationCharacterization_001`	

Table 165 lists the test case description for 'TC_EVCC_AC_VTB_AttenuationCharacterization_002'.

Table 165 — Test case description for 'TC_EVCC_AC_VTB_AttenuationCharacterization_002'

TC Id	`TC_EVCC_AC_VTB_AttenuationCharacterization_002`
Test objective	Test System executes GoodCase procedure, changes the duty cycle to 96 % before sending a CM_SLAC_PARM.CNF message with the current runID, EV MAC and all additional valid parameters. Test System then waits for the CM_START_ATTEN_CHAR.IND and CM_MNBC_SOUND.IND messages so that the Test System can measure the individual attenuation values. This profile containing 58 attenuation entries will be send by CM_ATTEN_CHAR.IND message with the current runID, EV MAC and all additional valid parameters to the SUT. Test System then checks that 3 CM_START_ATTEN_CHAR.IND messages with the current runID and all additional valid parameters and 10 CM_MNBC_SOUND.IND messages with the current runID, a decrementing counter and all additional valid parameters are sent by the SUT. Furthermore the SUT shall confirm the attenuation values by sending the CM_ATTEN_CHAR.RSP message with the current runID, EV MAC and all additional valid parameters.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.1; 15118-3:A.9.2.3.2; 15118-3:6.4.3.2; 15118-3:A.9
Referenced requirement(s)	[V2G3-A09-18], [V2G3-A09-23], [V2G3-A09-25], [V2G3-A09-28], [V2G3-A09-37], [V2G3-M06-15], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002

PICS selection	`PICS_CMN_CMN_ChargingMode := aC,`
	`PICS_CMN_CMN_IdentificationMode := eIM`

PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmSlacParm_001`	
Expected behavior	
`f_EVCC_AC_TB_VTB_AttenuationCharacterization_001`	

8.4.3 EVCC test cases for CmValidate

Table 166 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_001'.

Table 166 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_001'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_001`
Test objective	Test System executes GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters and waits for the CM_VALIDATE.REQ message (step 2) so that the Test System can count the BCB toggles. The number of toggles will be send by CM_VALIDATE.CNF message (step 2) with 'result' equals to '02'H and all additional valid parameters to the SUT. Test System then checks that the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters is sent by the SUT. Furthermore the Test System checks that the CM_VALIDATE.REQ message (step 2) with a valid 'pilotTimer' unequals to '00'H and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-06], [V2G3-M09-07], [V2G3-M09-11], [V2G3-M09-12], [V2G3-M09-14], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-67], [V2G3-A09-68], [V2G3-M08-01], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-59], [V2G3-A09-60], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_001`	

Table 167 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_002'.

Table 167 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_002'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_002`
Test objective	Test System executes GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then changes the duty cycle to 10 % before sending a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters. Test System then waits for the CM_VALIDATE.REQ message (step 2) so that it can count the BCB toggles. The number of toggles will be send by CM_VALIDATE.CNF message (step 2) with 'result' equals to '02'H and all additional valid parameters to the SUT. Test System then checks that the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters is sent by the SUT. Furthermore the Test System checks that the CM_VALIDATE.REQ message (step 2) with a valid 'pilotTimer' unequals to '00'H and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-06], [V2G3-M09-07], [V2G3-M09-11], [V2G3-M09-12], [V2G3-M09-14], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-67], [V2G3-A09-68], [V2G3-M09-15], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-59], [V2G3-A09-60], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002

PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_001`	

Table 168 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_003'.

Table 168 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_003'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_003`
Test objective	Test System executes GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then changes the duty cycle to 96 % before sending a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters. Test System then waits for the CM_VALIDATE.REQ message (step 2) so that it can count the BCB toggles. The number of toggles will be send by CM_VALIDATE.CNF message (step 2) with 'result' equals to '02'H and all additional valid parameters to the SUT. Test System then checks that the CM_VALIDATE.REQ message (step 1) with 'pilotTimer' equals to '00'H and all additional valid parameters is sent by the SUT. Furthermore the Test System checks that the CM_VALIDATE.REQ message (step 2) with a valid 'pilotTimer' unequals to '00'H and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-06], [V2G3-M09-07], [V2G3-M09-11], [V2G3-M09-12], [V2G3-M09-14], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-67], [V2G3-A09-68], [V2G3-M09-15], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-59], [V2G3-A09-60], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_001`	

Table 169 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_004'.

Table 169 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_004'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_004`
Test objective	Test System executes GoodCase procedure, waits for the CM_VALIDATE.REQ message (step 1) and counts the number of CM_VALIDATE.REQ repetitions including 'pilotTimer' equals to '00'H and all additional valid parameters without sending a CM_VALIDATE.CNF message until the TT_match_response timer has expired. Test System then checks the repetition of CM_VALIDATE.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1

Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-62], [V2G3-A08-01], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_002`	

Table 170 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_005'.

Table 170 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_005'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_005`
Test objective	Test System executes GoodCase procedure, waits for the CM_VALIDATE.REQ message (step 1) and counts the number of CM_VALIDATE.REQ repetitions including 'pilotTimer' equals to '00'H and all additional valid parameters after sending an invalid 'signalType' equals to 'FF'H in the CM_VALIDATE.CNF messages. Test System then checks the repetition of CM_VALIDATE.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-62], [V2G3-A08-01], [V2G3-A09-64], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_003`	

Table 171 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_006'.

Table 171 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_006'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_006`
Test objective	Test System executes GoodCase procedure, waits for the CM_VALIDATE.REQ message (step 1) and counts the number of CM_VALIDATE.REQ repetitions including 'toggleNum' equals to '00'H and all additional valid parameters after sending an invalid 'signalType' equals to 'FF'H in the CM_VALIDATE.CNF messages. Test System then checks the repetition of CM_VALIDATE.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1

Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-62], [V2G3-A08-01], [V2G3-A09-64], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_003`	

Table 172 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_007'.

Table 172 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_007'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_007`
Test objective	Test System starts GoodCase procedure twice in parallel but the first instance sends no CM_VALIDATE.CNF message (step 1) until the TT_match_response timer has expired after receipt of a valid CM_VALIDATE.REQ message (step 1). Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-62], [V2G3-A08-01], [V2G3-A09-63], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_009, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidate_002`	

Table 173 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_008'.

Table 173 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_008'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_008`
Test objective	Test System starts GoodCase procedure twice in parallel but the first instance sends an invalid 'result' equals to '02'H in the CM_VALIDATE.CNF message (step 1) after receipt of valid CM_VALIDATE.REQ message (step 1). Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-

	3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-65], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_010, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidate_004`	

Table 174 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_009'.

Table 174 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_009'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_009`
Test objective	Test System starts GoodCase procedure twice in parallel, waiting for the CM_VALIDATE.REQ message (step 1). The first instance then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters and waits for the CM_VALIDATE.REQ message (step 2) so that this instance can count the BCB toggles. After execution of the BCB toggle sequence, the first instance waits until the TT_match_response timer has expired. Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.8; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A08-01], [V2G3-A09-54], [V2G3-A09-70], [V2G3-A09-71], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-59], [V2G3-A09-60], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_011, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidate_005`	

Table 175 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_010'.

Table 175 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_010'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_010`
Test objective	Test System starts GoodCase procedure twice in parallel, waiting for the CM_VALIDATE.REQ message (step 1). The first instance then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid

	parameters and waits for the CM_VALIDATE.REQ message (step 2) so that this instance can count the BCB toggles. After execution of the BCB toggle sequence, the first instance sends an invalid 'result' equals to '03'H in the CM_VALIDATE.CNF message (step 2). Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-72], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-59], [V2G3-A09-60], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_012, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidate_006`	

Table 176 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_011'.

Table 176 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_011'

TC Id	`TC_EVCC_CMN_VTB_CmValidate_011`
Test objective	Test System starts GoodCase procedure twice in parallel, waiting for the CM_VALIDATE.REQ message (step 1). The first instance then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters and waits for the CM_VALIDATE.REQ message (step 2) so that this instance can count the BCB toggles. After execution of the BCB toggle sequence, the first instance sends an invalid 'result' equals to '00'H in the CM_VALIDATE.CNF message (step 2). Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-72], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-59], [V2G3-A09-60], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := cmValidate`
PreCondition	
`f_EVCC_CMN_PR_DutyCycle_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmValidate_012, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001,`	

f_EVCC_CMN_TB_VTB_CmValidate_006

Table 177 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_012'.

Table 177 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_012'

TC Id	TC_EVCC_CMN_VTB_CmValidate_012
Test objective	Test System starts GoodCase procedure twice in parallel, waiting for the CM_VALIDATE.REQ message (step 1). The first instance then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters and waits for the CM_VALIDATE.REQ message (step 2) so that this instance can count the BCB toggles. After execution of the BCB toggle sequence, the first instance sends an invalid 'result' equals to '04'H in the CM_VALIDATE.CNF message (step 2). Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-72], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-59], [V2G3-A09-60], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmValidate_012, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidate_006	

Table 178 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_013'.

Table 178 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_013'

TC Id	TC_EVCC_CMN_VTB_CmValidate_013
Test objective	Test System starts GoodCase procedure twice in parallel but the first instance sends an invalid 'result' equals to '03'H in the CM_VALIDATE.CNF message (step 1) after receipt of valid CM_VALIDATE.REQ message (step 1). Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-56], [V2G3-A09-57], [V2G3-M09-08], [V2G3-A09-51], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate, PIXIT_EVCC_CMN_FallbackValidationFailed := continue_

PreCondition
f_EVCC_CMN_PR_DutyCycle_001
Expected behavior
f_EVCC_CMN_TB_VTB_CmValidate_010, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidate_004

Table 179 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_014'.

Table 179 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_014'

TC Id	TC_EVCC_CMN_VTB_CmValidate_014
Test objective	Test System starts GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '03'H and all additional valid parameters. Test System then checks if the SUT will stop the SLAC validation process by detection of TT_match_sequence timeout.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-51], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate, PIXIT_EVCC_CMN_FallbackValidationFailed := terminate
PreCondition	
f_EVCC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmValidate_004	

Table 180 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_015'.

Table 180 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_015'

TC Id	TC_EVCC_CMN_VTB_CmValidate_015
Test objective	Test System starts GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '03'H and all additional valid parameters. Test System then checks if the SUT will skip the SLAC validation process and continue the matching process by sending a SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-56], [V2G3-A09-57], [V2G3-M09-08], [V2G3-A09-51], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate, PIXIT_EVCC_CMN_FallbackValidationFailed := skip

PreCondition
f_EVCC_CMN_PR_AttenuationCharacterization_001
Expected behavior
f_EVCC_CMN_TB_VTB_CmValidate_004, f_EVCC_CMN_TB_VTB_CmSlacMatch_001

Table 181 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_016'.

Table 181 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_016'

TC Id	TC_EVCC_CMN_VTB_CmValidate_016
Test objective	Test System starts GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '04'H and all additional valid parameters. Test System then checks if the SUT will continue the SLAC validation process by sending a CM_VALIDATE.REQ message (step 2) with a valid 'pilotTimer' unequals to '00'H and all additional valid parameters.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-50], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate, PIXIT_EVCC_CMN_FallbackValidationNotRequired := continue_
PreCondition	
f_EVCC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmValidate_007	

Table 182 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_017'.

Table 182 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_017'

TC Id	TC_EVCC_CMN_VTB_CmValidate_017
Test objective	Test System starts GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '04'H and all additional valid parameters. Test System then checks if the SUT will skip the SLAC validation process and continue the matching process by sending a SLAC_MATCH.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-50], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate, PIXIT_EVCC_CMN_FallbackValidationNotRequired := skip

PreCondition
f_EVCC_CMN_PR_AttenuationCharacterization_001
Expected behavior
f_EVCC_CMN_TB_VTB_CmValidate_004, f_EVCC_CMN_TB_VTB_CmSlacMatch_001

Table 183 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_018'.

Table 183 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_018'

TC Id	TC_EVCC_CMN_VTB_CmValidate_018
Test objective	Test System starts GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '00'H and all additional valid parameters. Test System then checks if the SUT will retry the SLAC validation process after waiting for 'PIXIT_EVCC_CMN_ValidationRetry' seconds by sending a new CM_VALIDATE.REQ message (step 1).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-66], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	

PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate,
	PIXIT_EVCC_CMN_ConcurrentValidation := retry,
	PIXIT_EVCC_CMN_ValidationRetry

PreCondition	
f_EVCC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmValidate_008	

Table 184 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_019'.

Table 184 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_019'

TC Id	TC_EVCC_CMN_VTB_CmValidate_019
Test objective	Test System starts GoodCase procedure twice in parallel but the first instance sends a 'result' equals to '00'H in the CM_VALIDATE.CNF message (step 1) after receipt of valid CM_VALIDATE.REQ message (step 1). Test System then checks if the SUT will stop the SLAC validation process with the first instance and continue the SLAC validation process with the next potential EVSE (second instance).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.4; 15118-3:A.9.3.1; 15118-3:A.9.3.2; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-M09-07], [V2G3-M09-12], [V2G3-A09-52], [V2G3-A09-54], [V2G3-A09-56], [V2G3-A09-57], [V2G3-A09-66], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	

PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate,
	PIXIT_EVCC_CMN_ConcurrentValidation := iterate

PreCondition
f_EVCC_CMN_PR_DutyCycle_001
Expected behavior
f_EVCC_CMN_TB_VTB_CmValidate_010, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidate_004

Table 185 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_020'.

Table 185 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_020'

TC Id	TC_EVCC_CMN_VTB_CmValidate_020
Test objective	Test System executes GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then signals CP State E and sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters. Test System then checks that no CM_VALIDATE.REQ message (step 2) is sent by the SUT until the TT_match_sequence timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-127]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate
PreCondition	
f_EVCC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmValidate_013	

Table 186 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidate_021'.

Table 186 — Test case description for 'TC_EVCC_CMN_VTB_CmValidate_021'

TC Id	TC_EVCC_CMN_VTB_CmValidate_021
Test objective	Test System executes GoodCase procedure and waits for the CM_VALIDATE.REQ message (step 1). Test System then signals CP State F and sends a CM_VALIDATE.CNF message (step 1) with 'result' equals to '01'H and all additional valid parameters. Test System then checks that no CM_VALIDATE.REQ message (step 2) is sent by the SUT until the TT_match_sequence timer has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-127]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate
PreCondition	
f_EVCC_CMN_PR_AttenuationCharacterization_001	

Expected behavior
f_EVCC_CMN_TB_VTB_CmValidate_013

8.4.4 EVCC test cases for CmValidateOrCmSlacMatch

Table 187 lists the test case description for 'TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001'.

Table 187 — Test case description for 'TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001'

TC Id	TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001
Test objective	Test System executes GoodCase procedure and waits for CM_SLAC_Match.REQ message. Test System then checks that a CM_SLAC_Match.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters is sent by the SUT. If a CM_VALIDATE.REQ message was received before, a SLAC validation process shall be executed by SUT and Test System previously.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := unknown
PreCondition	
f_EVCC_CMN_PR_AttenuationCharacterization_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmValidateOrCmSlacMatch_001	

8.4.5 EVCC test cases for CmSlacMatch

Table 188 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_001'.

Table 188 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_001'

TC Id	TC_EVCC_CMN_VTB_CmSlacMatch_001
Test objective	Test System executes GoodCase procedure and waits for entering the SLAC validation process. After counting the BCB toggles, the Test System reports the number of toggles as part of the CM_VALIDATE.CNF message. The Test System then checks that a CM_SLAC_Match.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9.3.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A09-73], [V2G3-A09-74], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	PIXIT_EVCC_CMN_CmValidate := cmValidate
PreCondition	
f_EVCC_CMN_PR_CmValidate_001	

Expected behavior
`f_EVCC_CMN_TB_VTB_CmSlacMatch_001`

Table 189 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_002'.

Table 189 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_002'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_002`
Test objective	Test System executes GoodCase procedure without entering the SLAC validation process. The Test System then checks that a CM_SLAC_Match.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters is sent by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	`PIXIT_EVCC_CMN_CmValidate := none_`
PreCondition	
`f_EVCC_CMN_PR_AttenuationCharacterization_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacMatch_001`	

Table 190 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_003'.

Table 190 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_003'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_003`
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter without sending a CM_SLAC_MATCH.CNF message until the TT_match_response timer has expired. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacMatch_002`	

Table 191 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_004'.

Table 191 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_004'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_004`
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'applicationType' equals to 'FF'H in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacMatch_003`	

Table 192 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_005'.

Table 192 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_005'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_005`
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'securityType' equals to 'FF'H in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacMatch_003`	

Table 193 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_006'.

Table 193 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_006'

TC Id	TC_EVCC_CMN_VTB_CmSlacMatch_006
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'mvfLength' equals to '0000'H in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacMatch_003	

Table 194 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_007'.

Table 194 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_007'

TC Id	TC_EVCC_CMN_VTB_CmSlacMatch_007
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'evID' equals to '00000000000000000000000000000001'H in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmSlacMatch_003	

Table 195 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_008'.

Table 195 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_008'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_008`
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'evMac' equals to '000000000000'H in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacMatch_003`	

Table 196 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_009'.

Table 196 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_009'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_009`
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'evseID' equals to '0000000000000000000000000000000001'H in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacMatch_003`	

Table 197 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_010'.

Table 197 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_010'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_010`

Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'evseMac' equals to '000000000000'H in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmSlacMatch_003`	

Table 198 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_011'.

Table 198 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_011'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_011`
Test objective	Test System executes GoodCase procedure and counts the number of CM_SLAC_MATCH.REQ repetitions including the current runID, EV MAC, EVSE MAC and all additional valid parameter after sending an invalid 'runID' in CM_SLAC_MATCH.CNF after each CM_SLAC_MATCH.REQ message. Test System then checks the repetition of CM_SLAC_MATCH.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.8; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A08-01], [V2G3-A09-94], [V2G3-A09-95], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_randomHexStringGen, f_EVCC_CMN_TB_VTB_CmSlacMatch_003`	

Table 199 lists the test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_012'.

Table 199 — Test case description for 'TC_EVCC_CMN_VTB_CmSlacMatch_012'

TC Id	`TC_EVCC_CMN_VTB_CmSlacMatch_012`
Test objective	Test System starts GoodCase procedure twice in parallel but in the second instance the

	CM_ATTEN_CHAR.IND message will not be send. Test System then checks that a CM_SLAC_Match.REQ message with the current runID, EV MAC, EVSE MAC and all additional valid parameters is sent by the SUT if the TT_EV_atten_results timer expires and not all anticipated responses are received.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.2.3.2; 15118-3:A.9.3.1; 15118-3:A.9.4.1; 15118-3:A.9.4.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-38], [V2G3-A09-52], [V2G3-A09-91], [V2G3-A09-34], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_DutyCycle_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_AttenuationCharacterization_003, f_EVCC_CMN_TB_VTB_CmSlacParm_001, f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001, f_EVCC_CMN_TB_VTB_CmValidateOrCmSlacMatch_001	

8.4.6 EVCC test cases for PLCLinkStatus

8.4.6.1 Common test cases

Table 200 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_001'.

Table 200 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_001'

TC Id	TC_EVCC_CMN_VTB_PLCLinkStatus_001
Test objective	Test System executes GoodCase procedure, sends a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameter and then waits for AVLN establishment by exchange of communication with the internal PLC Node. Test System then checks that the data link connection is established within 'TT_match_join'.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:A.9.5.3.2
Referenced requirement(s)	[V2G3-A09-101]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_PLCLinkStatus_001	

Table 201 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_002'.

Table 201 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_002'

TC Id	`TC_EVCC_CMN_VTB_PLCLinkStatus_002`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Test System then checks that the SUT leaves the logical network within 'TP_match_leave' if CP State E was detected before.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:9.7
Referenced requirement(s)	[V2G3-M09-19]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_002`	

Table 202 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_003'.

Table 202 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_003'

TC Id	`TC_EVCC_CMN_VTB_PLCLinkStatus_003`
Test objective	Test System executes GoodCase procedure, signals CP State E and sends a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameters. Test System then checks that no data link connection is established until the TT_match_join timer has expired.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-127]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_003`	

Table 203 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_004'.

Table 203 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_004'

TC Id	`TC_EVCC_CMN_VTB_PLCLinkStatus_004`
Test objective	Test System executes GoodCase procedure, signals CP State F and sends a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameters. Test System then checks that no data link connection is established until the TT_match_join timer has expired.

Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:A.9.8
Referenced requirement(s)	[V2G3-A09-127]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_003`	

Table 204 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_005'.

Table 204 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_005'

TC Id	`TC_EVCC_CMN_VTB_PLCLinkStatus_005`
Test objective	Test System executes GoodCase procedure (SECC delay for signalling a 5 % duty cycle was set to 7,5 s), sends a CM_SLAC_MATCH.CNF message with the current runID, valid NID and NMK, EV MAC, EVSE MAC, and all additional valid parameter and then waits for AVLN establishment by exchange of communication with the internal PLC Node. Test System then checks that the data link connection is established within 'TT_match_join'.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:A.9.5.3.2
Referenced requirement(s)	[V2G3-A09-101]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_001`	

Table 205 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_006'.

Table 205 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_006'

TC Id	`TC_EVCC_CMN_VTB_PLCLinkStatus_006`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT starts a new matching process if Test System performs a CP State X2 to X1 to E to 5 % duty cycle transition.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.2.1
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-13]
Config Id	CF_05_002

PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_006`	

Table 206 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_007'.

Table 206 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_007'

TC Id	`TC_EVCC_CMN_VTB_PLCLinkStatus_007`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT starts a new matching process if Test System performs a CP State X2 to X1 to F to 5 % duty cycle transition.
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.2.1
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-13]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_006`	

Table 207 lists the test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_008'.

Table 207 — Test case description for 'TC_EVCC_CMN_VTB_PLCLinkStatus_008'

TC Id	`TC_EVCC_CMN_VTB_PLCLinkStatus_008`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT leaves the logical network (setting previous key again on Test System side).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.7; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-34], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	
PIXIT selection	
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	

f_EVCC_CMN_TB_VTB_PLCLinkStatus_008

8.4.6.2 AC specific test cases

Table 208 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_001'.

Table 208 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_001'

TC Id	TC_EVCC_AC_VTB_PLCLinkStatus_001
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by initiating a oscillator B1/B2 transition and waits for successful data link detection triggered by the SUT. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (B1/B2 detection), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-22], [V2G3-M07-23], [V2G3-M07-29], [V2G3-M07-30], [V2G3-M09-16], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	PICS_CMN_CMN_ChargingMode := aC, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true
PIXIT selection	PIXIT_EVCC_CMN_Pause := pause
PreCondition	
f_EVCC_AC_PR_SessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase	
Expected behavior	
f_EVCC_CMN_TB_VTB_PLCLinkStatus_004	

Table 209 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_002'.

Table 209 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_002'

TC Id	TC_EVCC_AC_VTB_PLCLinkStatus_002
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PIXIT_CMN_CMN_WakeUp' the SUT resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (initiating a BCB toggle), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-28], [V2G3-M07-29], [V2G3-M09-16], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	PICS_CMN_CMN_ChargingMode := aC, PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true

PIXIT selection	`PIXIT_EVCC_CMN_Pause := pause`
PreCondition	
`f_EVCC_AC_PR_SessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_005`	

Table 210 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_003'.

Table 210 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_003'

TC Id	`TC_EVCC_AC_VTB_PLCLinkStatus_003`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT starts a new matching process if Test System performs a CP State X2 to X1 to E to nominal duty cycle transition (Option A).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.2.2
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-14], [V2G3-M07-15]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := aC`
PIXIT selection	`PIXIT_EVCC_AC_ConnectionLossHandling := optionA`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_006`	

Table 211 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_004'.

Table 211 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_004'

TC Id	`TC_EVCC_AC_VTB_PLCLinkStatus_004`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT starts a new matching process after 'T_conn_resetup' (Option B).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.2.2
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-14], [V2G3-M07-16], [V2G3-M07-17]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := aC`

PIXIT selection	`PIXIT_EVCC_AC_ConnectionLossHandling := optionB,`
	`PIXIT_EVCC_AC_TconnResetup`

PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	

f_EVCC_AC_TB_VTB_PLCLinkStatus_001

Table 212 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_005'.

Table 212 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_005'

TC Id	TC_EVCC_AC_VTB_PLCLinkStatus_005
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT starts a new matching process if Test System performs a CP State X2 to X1 to E to nominal duty cycle transition during reinit phase (Option B).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.2.2
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-14], [V2G3-M07-16], [V2G3-M07-17], [V2G3-M07-18]
Config Id	CF_05_002
PICS selection	PICS_CMN_CMN_ChargingMode := aC
PIXIT selection	PIXIT_EVCC_AC_ConnectionLossHandling := optionB, PIXIT_EVCC_AC_TconnResetup
PreCondition	
f_EVCC_CMN_PR_PLCLinkStatus_001	
Expected behavior	
f_EVCC_AC_TB_VTB_PLCLinkStatus_002	

Table 213 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_006'.

Table 213 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_006'

TC Id	TC_EVCC_AC_VTB_PLCLinkStatus_006
Test objective	Test System executes GoodCase procedure and waits for a terminated V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Terminate' and all additional mandatory parameters. Furthermore Test System checks if the SUT terminates the TCP connection after receipt of a valid SessionStopRes message. Test System then checks that the SUT leaves the logical network after 'TP_match_leave' by using the CM_NW_STATS message sequence.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.7; 15118-3:A.9.7
Referenced requirement(s)	[V2G3-M09-17], [V2G3-A09-121]
Config Id	CF_05_002
PICS selection	PICS_CMN_CMN_ChargingMode := aC, PICS_CMN_CMN_CombinedTesting := true
PIXIT selection	PIXIT_EVCC_CMN_Pause := none_
PreCondition	
f_EVCC_AC_PR_SessionStop_003	
Expected behavior	
f_EVCC_CMN_TB_VTB_PLCLinkStatus_007	

Table 214 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_007'.

Table 214 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_007'

TC Id	`TC_EVCC_AC_VTB_PLCLinkStatus_007`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT starts a new matching process if Test System performs a CP State X2 to X1 to F to nominal duty cycle transition (Option A).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.2.2
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-14], [V2G3-M07-15]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := aC`
PIXIT selection	`PIXIT_EVCC_AC_ConnectionLossHandling := optionA`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_006`	

Table 215 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_008'.

Table 215 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_008'

TC Id	`TC_EVCC_AC_VTB_PLCLinkStatus_008`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Afterwards Test System initiates a connection loss by setting a new key at the local node. Test System then checks that the SUT starts a new matching process if Test System performs a CP State X2 to X1 to F to nominal duty cycle transition during reinit phase (Option B).
Document reference	Document: ISO:15118-3:2015:IS; HPGP Spec 1.1 Section(s): 15118-3:7.5.1; 15118-3:7.5.2.2
Referenced requirement(s)	[V2G3-M07-03], [V2G3-M07-14], [V2G3-M07-16], [V2G3-M07-17], [V2G3-M07-18]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := aC`

PIXIT selection	`PIXIT_EVCC_AC_ConnectionLossHandling := optionB,`
	`PIXIT_EVCC_AC_TconnResetup`

PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_AC_TB_VTB_PLCLinkStatus_002`	

Table 216 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_009'.

Table 216 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_009'

TC Id	`TC_EVCC_AC_VTB_PLCLinkStatus_009`
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current

	SessionID, ChargingSession 'Pause' and all additional mandatory parameters (sending a PmaxScheduleList with the first Pmax element = 0W (par_SECC_Pmax0W) within ChargeParameterDiscoveryRes message). After 'par_SECC_Pmax0W' the SUT resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (initiating a BCB toggle), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-28], [V2G3-M07-29], [V2G3-M09-16], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true, PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepWithoutCharge`
PIXIT selection	
PreCondition	
`f_EVCC_AC_PR_SessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_005`	

Table 217 lists the test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_010'.

Table 217 — Test case description for 'TC_EVCC_AC_VTB_PLCLinkStatus_010'

TC Id	`TC_EVCC_AC_VTB_PLCLinkStatus_010`
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters (sending a PmaxScheduleList with the second Pmax element = 0W (par_SECC_Pmax0W) within ChargeParameterDiscoveryRes message). After 'par_SECC_Pmax0W' the SUT resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (initiating a BCB toggle), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-28], [V2G3-M07-29], [V2G3-M09-16], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := aC, par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true, PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepAfterCharge`
PIXIT selection	
PreCondition	
`f_EVCC_AC_PR_SessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_005`	

8.4.6.3 DC specific test cases

Table 218 lists the test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_001'.

Table 218 — Test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_001'

TC Id	TC_EVCC_DC_VTB_PLCLinkStatus_001
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PICS_CMN_CMN_WakeUp' Test System resumes the previously paused session by signalling 5 % duty cycle and waits for successful data link detection triggered by the SUT. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (5 % duty cycle detection), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-22], [V2G3-M07-23], [V2G3-M07-29], [V2G3-M07-30], [V2G3-M09-16], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	PICS_CMN_CMN_ChargingMode := dC, PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true
PIXIT selection	PIXIT_EVCC_CMN_Pause := pause
PreCondition	
f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase	
Expected behavior	
f_EVCC_CMN_TB_VTB_PLCLinkStatus_004	

Table 219 lists the test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_002'.

Table 219 — Test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_002'

TC Id	TC_EVCC_DC_VTB_PLCLinkStatus_002
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters. After 'PIXIT_CMN_CMN_WakeUp' the SUT resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (initiating a BCB toggle), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-28], [V2G3-M07-29], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	PICS_CMN_CMN_ChargingMode := dC, PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true

PIXIT selection	`PIXIT_EVCC_CMN_Pause := pause`
PreCondition	
`f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_005`	

Table 220 lists the test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_003'.

Table 220 — Test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_003'

TC Id	`TC_EVCC_DC_VTB_PLCLinkStatus_003`
Test objective	Test System executes GoodCase procedure and waits for a terminated V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Terminate' and all additional mandatory parameters. Furthermore Test System checks if the SUT terminates the TCP connection after receipt of a valid SessionStopRes message. Test System then checks that the SUT leaves the logical network after 'TP_match_leave' by using the CM_NW_STATS message sequence.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:9.7; 15118-3:A.9.7
Referenced requirement(s)	[V2G3-M09-17], [V2G3-A09-121]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := dC, PICS_CMN_CMN_CombinedTesting := true`
PIXIT selection	`PIXIT_EVCC_CMN_Pause := none_`
PreCondition	
`f_EVCC_DC_PR_WeldingDetectionOrSessionStop_003`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_007`	

Table 221 lists the test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_004'.

Table 221 — Test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_004'

TC Id	`TC_EVCC_DC_VTB_PLCLinkStatus_004`
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters (sending a PmaxScheduleList with the first Pmax element = 0W (par_SECC_Pmax0W) within ChargeParameterDiscoveryRes message). After 'par_SECC_Pmax0W' the SUT resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (initiating a BCB toggle), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-28], [V2G3-M07-29], [V2G3-M09-16], [V2G3-M12-01]

Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := dC, par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true, PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepWithoutCharge`
PIXIT selection	
PreCondition	
`f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_005`	

Table 222 lists the test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_005'.

Table 222 — Test case description for 'TC_EVCC_DC_VTB_PLCLinkStatus_005'

TC Id	`TC_EVCC_DC_VTB_PLCLinkStatus_005`
Test objective	Test System executes GoodCase procedure and waits for a paused V2G communication session initiated by the SUT by receiving a SessionStopReq message with the current SessionID, ChargingSession 'Pause' and all additional mandatory parameters (sending a PmaxScheduleList with the second Pmax element = 0W (par_SECC_Pmax0W) within ChargeParameterDiscoveryRes message). After 'par_SECC_Pmax0W' the SUT resumes the previously paused session by initiating a BCB toggle. Test System checks that the SUT signals CP State B during the sleeping phase. Furthermore Test System checks the wake-up process (initiating a BCB toggle), the successful data link detection within 'par_T_conn_resume' and the start of the SDP process by the SUT (SUT is ready for Binding process).
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:7.6.1; 15118-3:7.6.2; 15118-3:7.6.2.2; 15118-3:9.5; 15118-3:12.3
Referenced requirement(s)	[V2G3-M07-19], [V2G3-M07-21], [V2G3-M07-28], [V2G3-M07-29], [V2G3-M09-16], [V2G3-M12-01]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_ChargingMode := dC, par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp, PICS_CMN_CMN_CombinedTesting := true, PICS_EVCC_CMN_PmaxSchedulewithZeroPow := sleepAfterCharge`
PIXIT selection	
PreCondition	
`f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002, f_EVCC_setPwmMode, f_EVCC_startSleepingPhase`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_PLCLinkStatus_005`	

8.4.7 EVCC test cases for CmAmpMap

Table 223 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_001'.

Table 223 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_001'

TC Id	`TC_EVCC_CMN_VTB_CmAmpMap_001`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN and sends CM_AMP_MAP.REQ message with a new amplitude map and all additional valid parameters.

	Test System then checks that a CM_AMP_MAP.CNF message with 'result' equals to '00'H is sent by the SUT. Furthermore the reduction of the transmission power of the requested carriers will be checked with additional equipment.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2; 15118-3:A.9.5.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-101], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmAmpMap_001, f_EVCC_CMN_checkTXPowerLimitation`	

Table 224 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_002'.

Table 224 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_002'

TC Id	`TC_EVCC_CMN_VTB_CmAmpMap_002`
Test objective	Test System executes GoodCase procedure and establishes a new AVLN. Test System then checks that CM_AMP_MAP.REQ message with a new amplitude map and all additional valid parameters is sent by the SUT until 'par_TT_amp_map_exchange' has expired.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2; 15118-3:A.9.5.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-109], [V2G3-A09-111], [V2G3-A09-101], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := false`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmAmpMap_002`	

Table 225 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_003'.

Table 225 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_003'

TC Id	`TC_EVCC_CMN_VTB_CmAmpMap_003`
Test objective	Test System executes GoodCase procedure and counts the number of CM_AMP_MAP.REQ repetitions including a new amplitude map and all additional valid parameter without sending a CM_AMP_MAP.CNF message until the TT_match_response timer has expired. Test System then checks the repetition of CM_AMP_MAP.REQ messages and whether it is limited to 2 retries by the SUT.
Document	Document: ISO:15118-3:2015:IS

reference	Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-109], [V2G3-A09-111], [V2G3-A09-112]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := false`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmAmpMap_003`	

Table 226 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_004'.

Table 226 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_004'

TC Id	`TC_EVCC_CMN_VTB_CmAmpMap_004`
Test objective	Test System executes GoodCase procedure and counts the number of CM_AMP_MAP.REQ repetitions including a new amplitude map and all additional valid parameter after sending an invalid 'result' equals to 'FF'H in the CM_AMP_MAP.CNF messages. Test System then checks the repetition of CM_AMP_MAP.REQ messages and whether it is limited to 2 retries by the SUT.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-109], [V2G3-A09-111], [V2G3-A09-112], [V2G3-A09-114]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := false`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmAmpMap_004`	

Table 227 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_005'.

Table 227 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_005'

TC Id	`TC_EVCC_CMN_VTB_CmAmpMap_005`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN and sends an invalid CM_AMP_MAP.REQ message with 'amLen' equals to '00'H and all additional valid parameters. Test System then checks that no CM_AMP_MAP.CNF message is sent by the SUT until TT_match_response timer has expired. This sequence will be repeated for 2 retries.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-115], [V2G3-A09-113]

Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmAmpMap_005`	

Table 228 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_006'.

Table 228 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_006'

TC Id	`TC_EVCC_CMN_VTB_CmAmpMap_006`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN, sends a CM_AMP_MAP.REQ message with a new amplitude map and all additional valid parameters and waits for a valid CM_AMP_MAP.CNF message. Test System then sends another valid CM_AMP_MAP.REQ message and checks that a CM_AMP_MAP.CNF message with 'result' equals to '00'H and all additional valid parameters is sent by the SUT again.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-116]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`
PreCondition	
`f_EVCC_CMN_PR_PLCLinkStatus_001`	
Expected behavior	
`f_EVCC_CMN_TB_VTB_CmAmpMap_006`	

Table 229 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_007'.

Table 229 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_007'

TC Id	`TC_EVCC_CMN_VTB_CmAmpMap_007`
Test objective	Test System executes GoodCase procedure, establishes a new AVLN and sends a burst of 3 CM_AMP_MAP.REQ messages with a new amplitude map and all additional valid parameters. Test System then checks that a CM_AMP_MAP.CNF message with 'result' equals to '00'H and all additional valid parameters is sent by the SUT for each request message.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-116]
Config Id	CF_05_002
PICS selection	`PICS_CMN_CMN_InitiateCmAmpMap := true`
PIXIT selection	`PIXIT_CMN_CMN_CmAmpMap := true`

PreCondition
f_EVCC_CMN_PR_PLCLinkStatus_001
Expected behavior
f_EVCC_CMN_TB_VTB_CmAmpMap_007

Table 230 lists the test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_008'.

Table 230 — Test case description for 'TC_EVCC_CMN_VTB_CmAmpMap_008'

TC Id	TC_EVCC_CMN_VTB_CmAmpMap_008
Test objective	Test System executes GoodCase procedure, initiates an amplitude map process with transmission power limitation check. Test System then checks that the SUT sends a valid SDP request message. Furthermore the usability of the matched bandwidth (number of allocatable carriers) will be checked after amplitude map process.
Document reference	Document: ISO:15118-3:2015:IS Section(s): 15118-3:A.9.6.1; 15118-3:A.9.6.2; 15118-3:A.9.6.3.2; 15118-3:A.9.5.3.2; 15118-3:A.9; 15118-3:A.9.2.1
Referenced requirement(s)	[V2G3-A09-106], [V2G3-A09-110], [V2G3-A09-115], [V2G3-A09-101], [V2G3-A09-01], [V2G3-A09-17]
Config Id	CF_05_002
PICS selection	PICS_CMN_CMN_InitiateCmAmpMap := true
PIXIT selection	PIXIT_CMN_CMN_CmAmpMap := true
PreCondition	
f_EVCC_CMN_PR_CmAmpMap_001	
Expected behavior	
f_EVCC_CMN_TB_VTB_CmAmpMap_008	

Annex A
(normative)

Configuration specifications

A.1 Timer configuration

```
module Timer_15118 {

    group Timer_par_15118 {

        //  ------------------ Non-standardised CMN Processing timeouts -------
        modulepar float par_CMN_Transmission_Delay := 0.25;
        modulepar float par_CMN_HAL_Timeout := 2.0;
        modulepar float par_CMN_waitForNextHAL := 0.5;
        modulepar float par_CMN_Intern_PTC_Timeout := 0.4;
        modulepar float par_CMN_setKey := 3.0;

        // ------------------ Non-standardised EVCC Processing timeouts -------
        modulepar float par_EVCC_waitForNextTC := 2.0;
        // ------------------ Non-standardised SECC Processing timeouts -------

        modulepar float par_SECC_waitForNextTC := 20.0;
        modulepar float par_SECC_T_step_X1 := 3.0;

        modulepar float par_SECC_Pmax0W := 200.0;
    }
}


module Timer_15118_3 {

    import from DataStructure_SLAC all;

    group Timer_par_15118_3 {

        //  ------------------ SLAC timeout -----------------------------
        modulepar float par_TT_match_response := 0.2;
        modulepar float par_TP_match_sequence := 0.1;
        modulepar float par_TP_EV_batch_msg_interval := 0.03;
        modulepar float par_TT_EV_atten_results := 1.2;
        modulepar float par_TT_EVSE_SLAC_init_min := 20.0;
        modulepar float par_TT_EVSE_SLAC_init_max := 50.0;
        modulepar float par_TT_EVSE_match_MNBC := 0.6;
        modulepar float par_TT_EVSE_match_session := 10.0;
        modulepar float par_TT_match_sequence := 0.4;
        modulepar float par_TT_match_join := 12.0;
        modulepar float par_TT_link_status_response := 5.0;
        modulepar float par_TT_matching_rate := 0.4;
        modulepar float par_TT_matching_repetition := 10.0;
        modulepar float par_TP_match_response := 0.1;
        modulepar float par_T_step_EF_min := 4.0;
        modulepar float par_T_step_EF_max := 8.0;
        modulepar float par_T_conn_max_comm := 8.0;
        modulepar float par_TT_polling_pause := 0.25;
        modulepar float par_TP_match_leave := 1.0;
        modulepar float par_TP_EV_vald_state_duration := 0.3;
        modulepar float par_T_vald_state_duration_max := 0.4;
        modulepar float par_TP_EV_vald_toggle := 2.0;
        modulepar float par_TT_amp_map_exchange := 0.2;
        modulepar float par_T_conn_resume := 6.0;

        //  ------------------ Non-standardised CMN Processing timeouts --------
        modulepar float par_CMN_waitForKeyReset := 20.0;
        modulepar float par_CMN_waitForConnectionLoss := 5.0;

        // ------------------ Non-standardised SECC Processing timeouts --------
        modulepar float par_SECC_change_to_Nominal := 3.0;
        modulepar float par_SECC_EIM_Timeout := 10.0;
        modulepar float par_SECC_PLCNodeReady_delay := 1.0;
        modulepar float par_SECC_waitForEIMAuthorization := 10.0;
        modulepar float par_SECC_waitForPlugin := 600.0;
        // ------------------ Non-standardised EVCC Processing timeouts --------
        modulepar float par_EVCC_PLCNodeReady_delay := 15.0;

        modulepar float par_EVCC_setDC_delay := 1.0;
    }
}
```

A.2 PICS configuration

```
module Pics_15118 {

    import from DataStructure_PICS_15118 all;

    group PICS_15118 {


        group PICS_CMN {

            modulepar boolean PICS_CMN_CMN_CombinedTesting := false;

            modulepar ChargingMode PICS_CMN_CMN_ChargingMode := aC;

            modulepar IdentificationMode PICS_CMN_CMN_IdentificationMode := eIM;

            modulepar PlugType PICS_CMN_CMN_PlugType := type2;

            modulepar CableCapabilityACType PICS_CMN_AC_CableCapability := capability32A;

            modulepar float PICS_CMN_CMN_WakeUp := 300.0;
        }

        group PICS_SECC_Tester {

            modulepar boolean PICS_SECC_CMN_Pause := false;

            modulepar EIMDone PICS_SECC_CMN_EIMDone := v2gAuthorization;
        }

        group PICS_EVCC_Tester {


            modulepar ZeroPow PICS_EVCC_CMN_PmaxSchedulewithZeroPow := none_;
        }
    }
}


module Pics_15118_3 {

    import from DataStructure_PICS_15118_3 all;

    group PICS_15118_3 {


        group PICS_CMN {

            modulepar boolean PICS_CMN_CMN_InitiateCmAmpMap := true;

            modulepar boolean PICS_CMN_CMN_SlacTimeouts := true;

            modulepar boolean PICS_CMN_CMN_InvalidSlacDataFieldsAndMessages := true;


            modulepar boolean PICS_CMN_CMN_InvalidStatesAndDutyCycles := true;
        }
    }
}
```

A.3 PIXIT configuration

```
module Pixit_15118 {
    import from DataStructure_PIXIT_15118 all;

    group PIXIT_15118 {


        group PIXIT_CMN {

            modulepar float PIXIT_CMN_CMN_WakeUp := 200.0;
        }

        group PIXIT_EVCC_Tester {


            modulepar Pause PIXIT_EVCC_CMN_Pause := unknown;
        }
    }
}


module Pixit_15118_3 {

    import from DataStructure_PIXIT_15118_3 all;

    group PIXIT_15118_3 {

        group PIXIT_CMN_Tester {

            modulepar boolean PIXIT_CMN_CMN_CmAmpMap := false;
        }

        group PIXIT_SECC_Tester {

            modulepar CmValidateSECC PIXIT_SECC_CMN_CmValidate := none_;

            modulepar DutyCycle PIXIT_SECC_AC_InitialDutyCyle := dc5;

            modulepar boolean PIXIT_SECC_CMN_ArchitectureValidationNotRequired := false;

            modulepar CLHandling PIXIT_SECC_AC_ConnectionLossHandling := optionA;
        }

        group PIXIT_EVCC_Tester {

            modulepar CmValidateEVCC PIXIT_EVCC_CMN_CmValidate := unknown;

            modulepar ValidationFallbackHandling
                      PIXIT_EVCC_CMN_FallbackValidationFailed := unknown;

            modulepar ValidationFallbackHandling
                      PIXIT_EVCC_CMN_FallbackValidationNotRequired:= unknown;

            modulepar ConcurrentValidationHandling PIXIT_EVCC_CMN_ConcurrentValidation:= unknown;

            modulepar boolean PIXIT_EVCC_CMN_TTMatchingRepetitionConfig:= false;

            modulepar float PIXIT_EVCC_CMN_TTMatchingRepetition:= 10.0;

            modulepar float PIXIT_EVCC_CMN_TTMatchingRate:= 0.4;

            modulepar float PIXIT_EVCC_CMN_ValidationRetry:= 1.0;

            modulepar CLHandling PIXIT_EVCC_AC_ConnectionLossHandling := optionA;


            modulepar float PIXIT_EVCC_AC_TconnResetup := 15.0;
        }
    }
}
```

Annex B
(normative)

Control part specification

B.1 SECC control parts

B.1.1 AC specific control parts

```
module AC_SECC_SLAC_Error_Control {

    import from TestCases_SECC_CmSlacParm all;
    import from TestCases_SECC_AttenuationCharacterization all;
    import from TestCases_SECC_CmValidate all;
    import from TestCases_SECC_CmSlacMatch all;
    import from TestCases_SECC_PLCLinkStatus all;
    import from TestCases_SECC_CmAmpMap all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from Timer_15118 all;
    import from ComponentsAndPorts all;
    import from Pixit_15118 all;

    control {

        /** Test Group 1: SLAC timeouts **/

        if (PICS_CMN_CMN_SlacTimeouts) {

            // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_004())};

            if (PIXIT_SECC_AC_InitialDutyCyle == dc5 and
                PICS_CMN_CMN_IdentificationMode == eIM){
                execute(TC_SECC_AC_VTB_CmSlacParm_002())
            };

            // AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_004())};

            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_012())};

            // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
                execute(TC_SECC_CMN_VTB_CmValidate_003())
            };

            // CmSlacMatch +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_SECC_CMN_CmValidate == none_) {
                execute(TC_SECC_CMN_VTB_CmSlacMatch_005())
            };

            if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
                execute(TC_SECC_CMN_VTB_CmSlacMatch_006())
            };

            // CmAmpMap ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
                execute(TC_SECC_CMN_VTB_CmAmpMap_003())
            };
        }


        /** Test Group 2: Invalid SLAC data fields and messages **/

        if (PICS_CMN_CMN_InvalidSlacDataFieldsAndMessages) {

            // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_005())};
if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_006())};

// AttenuationCharacterization  +++++++++++++++++++++++++++++++++++++++++++++//
if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_005())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_006())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_007())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_008())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_009())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_010())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_011())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_013())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_014())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_015())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_016())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_017())};

if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_018())};

// CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
    execute(TC_SECC_CMN_VTB_CmValidate_004())
};

if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
    execute(TC_SECC_CMN_VTB_CmValidate_005())
};

if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
    execute(TC_SECC_CMN_VTB_CmValidate_006())
};

if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
    execute(TC_SECC_CMN_VTB_CmValidate_007())
};

if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
    execute(TC_SECC_CMN_VTB_CmValidate_008())
};

if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
    execute(TC_SECC_CMN_VTB_CmValidate_009())
};

if (PIXIT_SECC_CMN_CmValidate == none_) {
    execute(TC_SECC_CMN_VTB_CmValidate_010())
};

// CmSlacMatch ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
if (PIXIT_SECC_CMN_CmValidate == none_) {
    execute(TC_SECC_CMN_VTB_CmSlacMatch_007())
};

if (PIXIT_SECC_CMN_CmValidate == none_) {
    execute(TC_SECC_CMN_VTB_CmSlacMatch_009())
};

if (PIXIT_SECC_CMN_CmValidate == none_) {
    execute(TC_SECC_CMN_VTB_CmSlacMatch_011())
};

if (PIXIT_SECC_CMN_CmValidate == none_) {
    execute(TC_SECC_CMN_VTB_CmSlacMatch_013())
};

if (PIXIT_SECC_CMN_CmValidate == none_) {
    execute(TC_SECC_CMN_VTB_CmSlacMatch_015())
};
```

```
        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_017())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_019())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_021())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_008())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_010())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_012())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_014())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_016())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_018())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_020())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_022())
        };

        // CmAmpMap +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_004())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_005())
        };
    }


    /** Test Group 3: Invalid states and duty cycles **/

    if (PICS_CMN_CMN_InvalidStatesAndDutyCycles) {

        // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_007())};

        // AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_019())};

        // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_013())
        };

        // CmSlacMatch +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_023())
        };
        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_024())
        };
```

```
            // PLCLinkStatus ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_PLCLinkStatus_002())};

            if (true) {execute(TC_SECC_CMN_VTB_PLCLinkStatus_005())};

            if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
                PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
                execute(TC_SECC_AC_VTB_PLCLinkStatus_004())
            };

            if(PIXIT_SECC_AC_InitialDutyCyle == dc5) {
               execute(TC_SECC_AC_VTB_PLCLinkStatus_005())
            };

            if(PIXIT_SECC_AC_InitialDutyCyle == dc5 and
               PIXIT_SECC_AC_ConnectionLossHandling == optionA and
               PICS_CMN_CMN_IdentificationMode == eIM and
               PICS_SECC_CMN_EIMDone == afterPlugin) {
               execute(TC_SECC_AC_VTB_PLCLinkStatus_006())
            };

            if(PIXIT_SECC_AC_InitialDutyCyle == dc5 and
               PIXIT_SECC_AC_ConnectionLossHandling == optionB and
               PICS_CMN_CMN_IdentificationMode == eIM and
               PICS_SECC_CMN_EIMDone == afterPlugin) {
               execute(TC_SECC_AC_VTB_PLCLinkStatus_007())
            };

            if(PIXIT_SECC_AC_InitialDutyCyle == dc100) {
               execute(TC_SECC_AC_VTB_PLCLinkStatus_011())
            };
        }
    }
}


module AC_SECC_SLAC_Control {

    import from TestCases_SECC_CmSlacParm all;
    import from TestCases_SECC_AttenuationCharacterization all;
    import from TestCases_SECC_CmValidate all;
    import from TestCases_SECC_CmSlacMatch all;
    import from TestCases_SECC_CmAmpMap all;
    import from TestCases_SECC_PLCLinkStatus all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from ComponentsAndPorts all;
    import from Timer_15118_3 all;
    import from Timer_15118 all;
    import from Pixit_15118 all;

    control {

            // CmSlacParm +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_001())};

            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_002())};

            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_003())};

            if (PIXIT_SECC_AC_InitialDutyCyle == dc5 and
                PICS_CMN_CMN_IdentificationMode == eIM and
                PICS_SECC_CMN_EIMDone == afterPlugin){
                execute(TC_SECC_AC_VTB_CmSlacParm_001())
            };

            if (PIXIT_SECC_AC_InitialDutyCyle == dc100 and
                PICS_CMN_CMN_IdentificationMode == eIM and
                PICS_SECC_CMN_EIMDone == afterPlugin){
                execute(TC_SECC_AC_VTB_CmSlacParm_003())
            };

            if (PICS_CMN_CMN_IdentificationMode == eIM and
                PICS_SECC_CMN_EIMDone == beforePlugin){
                execute(TC_SECC_AC_VTB_CmSlacParm_004())
            };
    if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_008())};

    if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_009())};

    // AttenuationCharacterization +++++++++++++++++++++++++++++++++++++++++++++//
    if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_001())};

    if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_002())};

    if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_003())};

    if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_020())};


    // CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
    if (PIXIT_SECC_CMN_CmValidate == cmValidate){
        execute(TC_SECC_CMN_VTB_CmValidate_001())};

    if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
        execute(TC_SECC_CMN_VTB_CmValidate_002())
    };

    if (PIXIT_SECC_CMN_CmValidate == cmValidate and
        PIXIT_SECC_CMN_ArchitectureValidationNotRequired) {
        execute(TC_SECC_CMN_VTB_CmValidate_011())
    };

    if (PIXIT_SECC_CMN_CmValidate == cmValidate and
        PIXIT_SECC_CMN_ArchitectureValidationNotRequired) {
        execute(TC_SECC_CMN_VTB_CmValidate_012())
    };


    // CmSlacMatch ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
    if (PIXIT_SECC_CMN_CmValidate == none_){
        execute(TC_SECC_CMN_VTB_CmSlacMatch_001())
    };

    if (PIXIT_SECC_CMN_CmValidate == none_) {
        execute(TC_SECC_CMN_VTB_CmSlacMatch_003())
    };

    if (PIXIT_SECC_CMN_CmValidate == cmValidate){
        execute(TC_SECC_CMN_VTB_CmSlacMatch_002())
    };

    if (PIXIT_SECC_CMN_CmValidate == cmValidate){
        execute(TC_SECC_CMN_VTB_CmSlacMatch_004())
    };

// PLCLinkStatus +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
if (not PIXIT_CMN_CMN_CmAmpMap){ execute(TC_SECC_CMN_VTB_PLCLinkStatus_001())};

if (true) {execute(TC_SECC_CMN_VTB_PLCLinkStatus_003())};

if (true) {execute(TC_SECC_CMN_VTB_PLCLinkStatus_004())};

if (PIXIT_SECC_AC_InitialDutyCyle == dc5 and
    PICS_CMN_CMN_IdentificationMode == eIM and
    PICS_SECC_CMN_EIMDone == duringSlac) {
    execute(TC_SECC_AC_VTB_PLCLinkStatus_001())
};

if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
    PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
    execute(TC_SECC_AC_VTB_PLCLinkStatus_002())
};

if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
    PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp) {
    execute(TC_SECC_AC_VTB_PLCLinkStatus_003())
};

if (not PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting) {
    execute(TC_SECC_AC_VTB_PLCLinkStatus_008())
};
```

```
        if (PIXIT_SECC_AC_InitialDutyCyle == dc100 and
            PICS_CMN_CMN_IdentificationMode == eIM and
            PICS_SECC_CMN_EIMDone == duringSlac) {
            execute(TC_SECC_AC_VTB_PLCLinkStatus_009())
        };

        if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
            PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
            execute(TC_SECC_AC_VTB_PLCLinkStatus_010())
        };

        if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
            PICS_CMN_CMN_WakeUp < par_SECC_T_step_X1 and
            PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
            execute(TC_SECC_AC_VTB_PLCLinkStatus_012())
        };

        // CmAmpMap +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_001())
        };

        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_002())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_006())
        };

        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_007())
        };

        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_008())
        };
    }
}
```

B.1.2 DC specific control parts

```
module DC_SECC_SLAC_Error_Control {

    import from TestCases_SECC_CmSlacParm all;
    import from TestCases_SECC_AttenuationCharacterization all;
    import from TestCases_SECC_CmValidate all;
    import from TestCases_SECC_CmSlacMatch all;
    import from TestCases_SECC_PLCLinkStatus all;
    import from TestCases_SECC_CmAmpMap all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from ComponentsAndPorts all;
    import from Pixit_15118 all;
    import from Timer_15118 all;

    control {

        /** Test Group 1: SLAC timeouts **/

        if (PICS_CMN_CMN_SlacTimeouts) {

            // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_004())};

            // AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_004())};

            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_012())};

            // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
                execute(TC_SECC_CMN_VTB_CmValidate_003())
            };

            // CmSlacMatch +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_SECC_CMN_CmValidate == none_) {
```

```
            execute(TC_SECC_CMN_VTB_CmSlacMatch_005())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_006())
        };

        // CmAmpMap ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_003())
        };
    }


    /** Test Group 2: Invalid SLAC data fields and messages **/

    if (PICS_CMN_CMN_InvalidSlacDataFieldsAndMessages) {

        // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_005())};

        if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_006())};

        // AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_005())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_006())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_007())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_008())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_009())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_010())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_011())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_013())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_014())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_015())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_016())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_017())};

        if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_018())};

        // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_004())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_005())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_006())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_007())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_008())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_009())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmValidate_010())
        };
        // CmSlacMatch ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_007())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_009())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_011())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_013())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_015())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_017())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_019())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_021())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_008())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_010())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_012())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_014())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_016())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_018())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_020())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_022())
        };

        // CmAmpMap +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_004())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_005())
        };
    }


    /** Test Group 3: Invalid states and duty cycles **/
```

```
        if (PICS_CMN_CMN_InvalidStatesAndDutyCycles) {

            // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_CmSlacParm_007())};

            // AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_019())};

            // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
                execute(TC_SECC_CMN_VTB_CmValidate_013())
            };

            // CmSlacMatch +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_SECC_CMN_CmValidate == none_) {
                execute(TC_SECC_CMN_VTB_CmSlacMatch_023())
            };
            if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
                execute(TC_SECC_CMN_VTB_CmSlacMatch_024())
            };

            // PLCLinkStatus +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_PLCLinkStatus_002())};

            if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
                PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
                execute(TC_SECC_DC_VTB_PLCLinkStatus_003())
            };

            if(true) {
               execute(TC_SECC_DC_VTB_PLCLinkStatus_004())
            };
        }
    }
}


module DC_SECC_SLAC_Control {

    import from TestCases_SECC_CmSlacParm all;
    import from TestCases_SECC_AttenuationCharacterization all;
    import from TestCases_SECC_CmValidate all;
    import from TestCases_SECC_CmSlacMatch all;
    import from TestCases_SECC_CmAmpMap all;
    import from TestCases_SECC_PLCLinkStatus all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from ComponentsAndPorts all;
    import from Timer_15118_3 all;
    import from Timer_15118 all;
    import from Pixit_15118 all;

    control {

            // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_001())};

            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_002())};

            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_003())};

            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_008())};

            if (true){ execute(TC_SECC_CMN_VTB_CmSlacParm_009())};


            // AttenuationCharacterization +++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_001())};

            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_002())};

            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_003())};

            if (true) {execute(TC_SECC_CMN_VTB_AttenuationCharacterization_020())};
        // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_SECC_CMN_CmValidate == cmValidate){
            execute(TC_SECC_CMN_VTB_CmValidate_001())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate) {
            execute(TC_SECC_CMN_VTB_CmValidate_002())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate and
            PIXIT_SECC_CMN_ArchitectureValidationNotRequired) {
            execute(TC_SECC_CMN_VTB_CmValidate_011())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate and
            PIXIT_SECC_CMN_ArchitectureValidationNotRequired) {
            execute(TC_SECC_CMN_VTB_CmValidate_012())
        };


        // CmSlacMatch  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_SECC_CMN_CmValidate == none_){
            execute(TC_SECC_CMN_VTB_CmSlacMatch_001())
        };

        if (PIXIT_SECC_CMN_CmValidate == none_) {
            execute(TC_SECC_CMN_VTB_CmSlacMatch_003())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate){
            execute(TC_SECC_CMN_VTB_CmSlacMatch_002())
        };

        if (PIXIT_SECC_CMN_CmValidate == cmValidate){
            execute(TC_SECC_CMN_VTB_CmSlacMatch_004())
        };

    // PLCLinkStatus +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
    if (not PIXIT_CMN_CMN_CmAmpMap){ execute(TC_SECC_CMN_VTB_PLCLinkStatus_001())};

    if (true) {execute(TC_SECC_CMN_VTB_PLCLinkStatus_003())};

    if (true) {execute(TC_SECC_CMN_VTB_PLCLinkStatus_004())};

    if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
        PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
        execute(TC_SECC_DC_VTB_PLCLinkStatus_001())
    };

    if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
        PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp) {
        execute(TC_SECC_DC_VTB_PLCLinkStatus_002())
    };

    if (not PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting) {
        execute(TC_SECC_DC_VTB_PLCLinkStatus_005())
    };

    if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
        PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
        execute(TC_SECC_DC_VTB_PLCLinkStatus_006())
    };

    if (PICS_SECC_CMN_Pause and PICS_CMN_CMN_CombinedTesting and
        PICS_CMN_CMN_WakeUp < par_SECC_T_step_X1 and
        PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp) {
        execute(TC_SECC_DC_VTB_PLCLinkStatus_007())
    };

    // CmAmpMap +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
    if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
        execute(TC_SECC_CMN_VTB_CmAmpMap_001())
    };

    if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
        execute(TC_SECC_CMN_VTB_CmAmpMap_002())
    };

    if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_006())
        };

        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_007())
        };

        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_SECC_CMN_VTB_CmAmpMap_008())
        };
    }
}
```

B.2 EVCC control parts

B.2.1 AC specific control parts

```
module AC_EVCC_SLAC_Error_Control {

    import from TestCases_EVCC_CmSlacParm all;
    import from TestCases_EVCC_AttenuationCharacterization all;
    import from TestCases_EVCC_CmValidate all;
    import from TestCases_EVCC_CmSlacMatch all;
    import from TestCases_EVCC_PLCLinkStatus all;
    import from TestCases_EVCC_CmAmpMap all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from ComponentsAndPorts all;

    control {

        /** Test Group 1: SLAC timeouts **/

        if (PICS_CMN_CMN_SlacTimeouts) {

            // CmSlacParm +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_002())};

            if (PIXIT_EVCC_CMN_TTMatchingRepetitionConfig) {
                execute(TC_EVCC_CMN_VTB_CmSlacParm_010())
            };

            // AttenuationCharacterization  +++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_003())};

            // CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_004())};

            if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_007())};

            if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
                execute(TC_EVCC_CMN_VTB_CmValidate_009())
            };

            // CmSlacMatch +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_003())};

            // CmAmpMap +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
                execute(TC_EVCC_CMN_VTB_CmAmpMap_003())
            };
        }


        /** Test Group 2: Invalid SLAC data fields and messages **/

        if (PICS_CMN_CMN_InvalidSlacDataFieldsAndMessages) {

            // CmSlacParm +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_003())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_004())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_005())};
if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_006())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_007())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_008())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_009())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_011())};

// AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_004())};

if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_005())};

if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_006())};

if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_007())};

if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_008())};

if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_009())};

if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_010())};

if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_011())};

// CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_005())};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_006())};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_008())};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
    execute(TC_EVCC_CMN_VTB_CmValidate_010())
};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
    execute(TC_EVCC_CMN_VTB_CmValidate_011())
};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
    execute(TC_EVCC_CMN_VTB_CmValidate_012())
};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
    PIXIT_EVCC_CMN_FallbackValidationFailed == continue_) {
    execute(TC_EVCC_CMN_VTB_CmValidate_013())
};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
    PIXIT_EVCC_CMN_FallbackValidationFailed == terminate) {
    execute(TC_EVCC_CMN_VTB_CmValidate_014())
};

if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
    PIXIT_EVCC_CMN_ConcurrentValidation == iterate) {
    execute(TC_EVCC_CMN_VTB_CmValidate_019())
};

// CmSlacMatch ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_004())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_005())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_006())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_007())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_008())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_009())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_010())};

if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_011())};
```

```
            // CmAmpMap ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
                execute(TC_EVCC_CMN_VTB_CmAmpMap_004())
            };

            if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
                execute(TC_EVCC_CMN_VTB_CmAmpMap_005())
            };
        }


        /** Test Group 3: Invalid states and duty cycles **/

        if (PICS_CMN_CMN_InvalidStatesAndDutyCycles) {

            // AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_012())};

            if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_013())};

            // CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
                execute(TC_EVCC_CMN_VTB_CmValidate_020())};

            if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
                execute(TC_EVCC_CMN_VTB_CmValidate_021())};

            // PLCLinkStatus  +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_002())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_003())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_004())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_006())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_007())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_008())};

            if (PIXIT_EVCC_AC_ConnectionLossHandling == optionA) {
                execute(TC_EVCC_AC_VTB_PLCLinkStatus_003())};

            if (PIXIT_EVCC_AC_ConnectionLossHandling == optionB) {
                execute(TC_EVCC_AC_VTB_PLCLinkStatus_004())};

            if (PIXIT_EVCC_AC_ConnectionLossHandling == optionB) {
                execute(TC_EVCC_AC_VTB_PLCLinkStatus_005())};

            if (PIXIT_EVCC_AC_ConnectionLossHandling == optionA) {
                execute(TC_EVCC_AC_VTB_PLCLinkStatus_007())};

            if (PIXIT_EVCC_AC_ConnectionLossHandling == optionB) {
                execute(TC_EVCC_AC_VTB_PLCLinkStatus_008())};
        }
    }
}


module AC_EVCC_SLAC_Control {

    import from TestCases_EVCC_CmSlacParm all;
    import from TestCases_EVCC_AttenuationCharacterization all;
    import from TestCases_EVCC_CmValidate all;
    import from TestCases_EVCC_CmSlacMatch all;
    import from TestCases_EVCC_PLCLinkStatus all;
    import from TestCases_EVCC_CmAmpMap all;
    import from TestCases_EVCC_CmValidateOrCmSlacMatch all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from Timer_15118_3 all;
    import from Timer_15118 all;
    import from ComponentsAndPorts all;
    import from Pixit_15118 all;
    import from Timer_15118_2 all;
```

```
control {

        // CmSlacParm +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_001())};

        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_012())};

        if (true){ execute(TC_EVCC_AC_VTB_CmSlacParm_001())};

        if (true){ execute(TC_EVCC_AC_VTB_CmSlacParm_002())};

        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_013())};

        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_014())};


        // AttenuationCharacterization +++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_001())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_002())};

        if (PICS_CMN_CMN_IdentificationMode == eIM) {
            execute(TC_EVCC_AC_VTB_AttenuationCharacterization_001())
        };

        if (PICS_CMN_CMN_IdentificationMode == eIM) {
            execute(TC_EVCC_AC_VTB_AttenuationCharacterization_002())
        };


        // CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_EVCC_CMN_CmValidate == cmValidate){
            execute(TC_EVCC_CMN_VTB_CmValidate_001())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_002())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_003())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_FallbackValidationFailed == skip) {
            execute(TC_EVCC_CMN_VTB_CmValidate_015())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_FallbackValidationNotRequired == continue_) {
            execute(TC_EVCC_CMN_VTB_CmValidate_016())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_FallbackValidationNotRequired == skip) {
            execute(TC_EVCC_CMN_VTB_CmValidate_017())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_ConcurrentValidation == retry) {
            execute(TC_EVCC_CMN_VTB_CmValidate_018())
        };


        // CmSlacMatch ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_EVCC_CMN_CmValidate == cmValidate){
            execute(TC_EVCC_CMN_VTB_CmSlacMatch_001())
        };

        if (PIXIT_EVCC_CMN_CmValidate == none_){
            execute(TC_EVCC_CMN_VTB_CmSlacMatch_002())
        };

        if (PIXIT_EVCC_CMN_CmValidate == unknown){
            execute(TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001())
        };

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_012())};
        // PLCLinkStatus +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (not PIXIT_CMN_CMN_CmAmpMap){ execute(TC_EVCC_CMN_VTB_PLCLinkStatus_001())};

        if (true){
            execute(TC_EVCC_CMN_VTB_PLCLinkStatus_005())};

        if (PIXIT_EVCC_CMN_Pause == pause and PICS_CMN_CMN_CombinedTesting and
            PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_AC_VTB_PLCLinkStatus_001())};

        if (PIXIT_EVCC_CMN_Pause == pause and PICS_CMN_CMN_CombinedTesting and
            PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_AC_VTB_PLCLinkStatus_002())};

        if (PIXIT_EVCC_CMN_Pause == none_ and PICS_CMN_CMN_CombinedTesting){
            execute(TC_EVCC_AC_VTB_PLCLinkStatus_006())};

        if (PICS_EVCC_CMN_PmaxSchedulewithZeroPow == sleepWithoutCharge and
            PICS_CMN_CMN_CombinedTesting and
            par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_AC_VTB_PLCLinkStatus_009())};

        if (PICS_EVCC_CMN_PmaxSchedulewithZeroPow == sleepAfterCharge and
            PICS_CMN_CMN_CombinedTesting and
            par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_AC_VTB_PLCLinkStatus_010())};

        // CmAmpMap ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_001())
        };

        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_002())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_006())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_007())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_008())
        };
    }
}
```

B.2.2 DC specific control parts

```
module DC_EVCC_SLAC_Error_Control {

    import from TestCases_EVCC_CmSlacParm all;
    import from TestCases_EVCC_AttenuationCharacterization all;
    import from TestCases_EVCC_CmValidate all;
    import from TestCases_EVCC_CmSlacMatch all;
    import from TestCases_EVCC_PLCLinkStatus all;
    import from TestCases_EVCC_CmAmpMap all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from ComponentsAndPorts all;

    control {

        /** Test Group 1: SLAC timeouts **/

        if (PICS_CMN_CMN_SlacTimeouts) {

            // CmSlacParm +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_002())};

            if (PIXIT_EVCC_CMN_TTMatchingRepetitionConfig) {
                execute(TC_EVCC_CMN_VTB_CmSlacParm_010())
        };

        // AttenuationCharacterization  +++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_003())};

        // CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_004())};

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_007())};

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_009())
        };

        // CmSlacMatch ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_003())};

        // CmAmpMap +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_003())
        };
    }

    /** Test Group 2: Invalid SLAC data fields and messages **/

    if (PICS_CMN_CMN_InvalidSlacDataFieldsAndMessages) {

        // CmSlacParm +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_003())};

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_004())};

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_005())};

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_006())};

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_007())};

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_008())};

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_009())};

        if (true) {execute(TC_EVCC_CMN_VTB_CmSlacParm_011())};

        // AttenuationCharacterization  +++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_004())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_005())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_006())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_007())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_008())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_009())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_010())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_011())};

        // CmValidate +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_005())};

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_006())};

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {execute(TC_EVCC_CMN_VTB_CmValidate_008())};

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_010())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_011())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_012())
            };

            if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
                PIXIT_EVCC_CMN_FallbackValidationFailed == continue_) {
                execute(TC_EVCC_CMN_VTB_CmValidate_013())
            };

            if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
                PIXIT_EVCC_CMN_FallbackValidationFailed == terminate) {
                execute(TC_EVCC_CMN_VTB_CmValidate_014())
            };

            if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
                PIXIT_EVCC_CMN_ConcurrentValidation == iterate) {
                execute(TC_EVCC_CMN_VTB_CmValidate_019())
            };

            // CmSlacMatch +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_004())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_005())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_006())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_007())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_008())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_009())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_010())};

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_011())};

            // CmAmpMap ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
                execute(TC_EVCC_CMN_VTB_CmAmpMap_004())
            };

            if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
                execute(TC_EVCC_CMN_VTB_CmAmpMap_005())
            };
        }


        /** Test Group 3: Invalid states and duty cycles **/

        if (PICS_CMN_CMN_InvalidStatesAndDutyCycles) {

            // AttenuationCharacterization  ++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_012())};

            if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_013())};

            // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
                execute(TC_EVCC_CMN_VTB_CmValidate_020())};

            if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
                execute(TC_EVCC_CMN_VTB_CmValidate_021())};

            // PLCLinkStatus  +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_002())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_003())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_004())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_006())};

            if (true) {execute(TC_EVCC_CMN_VTB_PLCLinkStatus_007())};
        }
    }
}


module DC_EVCC_SLAC_Control {
```

```
import from TestCases_EVCC_CmSlacParm all;
import from TestCases_EVCC_AttenuationCharacterization all;
import from TestCases_EVCC_CmValidate all;
import from TestCases_EVCC_CmSlacMatch all;
import from TestCases_EVCC_PLCLinkStatus all;
import from TestCases_EVCC_CmAmpMap all;
import from TestCases_EVCC_CmValidateOrCmSlacMatch all;
import from Pics_15118 all;
import from Pics_15118_3 all;
import from Pixit_15118_3 all;
import from Timer_15118_3 all;
import from ComponentsAndPorts all;
import from Timer_15118 all;
import from Pixit_15118 all;
import from Timer_15118_2 all;

control {

        // CmSlacParm ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_001())};

        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_012())};

        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_013())};

        if (true){ execute(TC_EVCC_CMN_VTB_CmSlacParm_014())};

        // AttenuationCharacterization ++++++++++++++++++++++++++++++++++++++++++++//
        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_001())};

        if (true) {execute(TC_EVCC_CMN_VTB_AttenuationCharacterization_002())};


        // CmValidate ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_EVCC_CMN_CmValidate == cmValidate){
            execute(TC_EVCC_CMN_VTB_CmValidate_001())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_002())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            execute(TC_EVCC_CMN_VTB_CmValidate_003())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_FallbackValidationFailed == skip) {
            execute(TC_EVCC_CMN_VTB_CmValidate_015())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_FallbackValidationNotRequired == continue_) {
            execute(TC_EVCC_CMN_VTB_CmValidate_016())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_FallbackValidationNotRequired == skip) {
            execute(TC_EVCC_CMN_VTB_CmValidate_017())
        };

        if (PIXIT_EVCC_CMN_CmValidate == cmValidate and
            PIXIT_EVCC_CMN_ConcurrentValidation == retry) {
            execute(TC_EVCC_CMN_VTB_CmValidate_018())
        };


        // CmSlacMatch +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PIXIT_EVCC_CMN_CmValidate == cmValidate){
            execute(TC_EVCC_CMN_VTB_CmSlacMatch_001())
        };

        if (PIXIT_EVCC_CMN_CmValidate == none_){
            execute(TC_EVCC_CMN_VTB_CmSlacMatch_002())
        };

        if (PIXIT_EVCC_CMN_CmValidate == unknown){
            execute(TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001())
            };

            if (true) {execute(TC_EVCC_CMN_VTB_CmSlacMatch_012())};

        // PLCLinkStatus +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (not PIXIT_CMN_CMN_CmAmpMap){ execute(TC_EVCC_CMN_VTB_PLCLinkStatus_001())};

        if (true){
            execute(TC_EVCC_CMN_VTB_PLCLinkStatus_005())};

        if (PIXIT_EVCC_CMN_Pause == pause and PICS_CMN_CMN_CombinedTesting and
            PIXIT_CMN_CMN_WakeUp > PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_DC_VTB_PLCLinkStatus_001())};

        if (PIXIT_EVCC_CMN_Pause == pause and PICS_CMN_CMN_CombinedTesting and
            PIXIT_CMN_CMN_WakeUp < PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_DC_VTB_PLCLinkStatus_002())};

        if (PIXIT_EVCC_CMN_Pause == none_ and PICS_CMN_CMN_CombinedTesting){
            execute(TC_EVCC_DC_VTB_PLCLinkStatus_003())};

        if (PICS_EVCC_CMN_PmaxSchedulewithZeroPow == sleepWithoutCharge and
            PICS_CMN_CMN_CombinedTesting and
            par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_DC_VTB_PLCLinkStatus_004())};

        if (PICS_EVCC_CMN_PmaxSchedulewithZeroPow == sleepAfterCharge and
            PICS_CMN_CMN_CombinedTesting and
            par_SECC_Pmax0W < PICS_CMN_CMN_WakeUp){
            execute(TC_EVCC_DC_VTB_PLCLinkStatus_005())};

        // CmAmpMap ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_001())
        };

        if (not PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_002())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_006())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_007())
        };

        if (PICS_CMN_CMN_InitiateCmAmpMap and PIXIT_CMN_CMN_CmAmpMap) {
            execute(TC_EVCC_CMN_VTB_CmAmpMap_008())
        };
    }
}
```

Annex C
(normative)

Test-case specifications for 15118-3

C.1 SECC + PLC bridge test cases

This subclause includes all test case *specifications* where the EVSE is defined as SUT.

C.1.1 SECC test cases for CmSlacParm

```
module TestCases_SECC_CmSlacParm {

    import from DataStructure_SLAC all;
    import from TestBehavior_SECC_CmSlacParm all;
    import from TestBehavior_SECC_AttenuationCharacterization all;
    import from TestBehavior_SECC_CmSlacMatch all;
    import from TestBehavior_SECC_CmSetKey all;
    import from TestBehavior_SECC_PLCLinkStatus all;
    import from TestBehavior_SECC_CmValidate all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_SECC_15118_3 all;
    import from PostConditions_SECC_15118_3 all;
    import from Timer_15118_3 all;
    import from Pixit_15118_3 all;
    import from Templates_CMN_SlacPayloadHeader all;

     testcase TC_SECC_CMN_VTB_CmSlacParm_001() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

     testcase TC_SECC_CMN_VTB_CmSlacParm_002() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        vc_state := C;
        preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

     testcase TC_SECC_CMN_VTB_CmSlacParm_003() runs on SECC_Tester system SystemSECC {
      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions---------------------------------------------------------
      f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      vc_state := D;
      preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

      //-------------- Test behavior----------------------------------------------------------
      if( preConVerdict == pass ) {
          f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions---------------------------------------------------------
      f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

  testcase TC_SECC_CMN_VTB_CmSlacParm_004() runs on SECC_Tester system SystemSECC {

     var HAL_61851_Listener v_HAL_61851_Listener;
     var verdicttype preConVerdict;

     // -------------- Pre Conditions---------------------------------------------------------
     f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

     //-------------- Test behavior----------------------------------------------------------
     if( preConVerdict == pass ) {
         f_SECC_CMN_TB_VTB_CmSlacParm_002(v_HAL_61851_Listener, omit ,false);
     } else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions---------------------------------------------------------
     f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

   testcase TC_SECC_CMN_VTB_CmSlacParm_005() runs on SECC_Tester system SystemSECC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions---------------------------------------------------------
      f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

      //-------------- Test behavior----------------------------------------------------------
      if( preConVerdict == pass ) {
          f_SECC_CMN_TB_VTB_CmSlacParm_002(v_HAL_61851_Listener,
                                  m_CMN_CMN_SlacPayloadHeaderInvalid_001(), true);
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions---------------------------------------------------------
      f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

   testcase TC_SECC_CMN_VTB_CmSlacParm_006() runs on SECC_Tester system SystemSECC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions---------------------------------------------------------
      f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

      //-------------- Test behavior----------------------------------------------------------
      if( preConVerdict == pass ) {
          f_SECC_CMN_TB_VTB_CmSlacParm_002(v_HAL_61851_Listener,
                                   m_CMN_CMN_SlacPayloadHeaderInvalid_002(), true);
      } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_CMN_VTB_CmSlacParm_007() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

    //-------------- Test behavior------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacParm_003(v_HAL_61851_Listener);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_CMN_VTB_CmSlacParm_008() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    vc_sleepAfterPlugOut := true;
    preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

    //-------------- Test behavior------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacParm_004();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_CMN_VTB_CmSlacParm_009() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    vc_sleepAfterPlugOut := true;
    preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

    //-------------- Test behavior------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacParm_005(v_HAL_61851_Listener);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_AC_VTB_CmSlacParm_001() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;
    var verdicttype verdict;
```

```
   // -------------- Pre Conditions----------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   vc_activateNominal := false;
   vc_testCaseSpecific := true;
   preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

   //-------------- Test behavior-----------------------------------------------------
   if( preConVerdict == pass ) {
       verdict := f_SECC_AC_TB_VTB_CmSlacParm_001(v_HAL_61851_Listener);
       if(verdict == pass) {
           verdict := f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
       }
       if(verdict == pass) {
           verdict := f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
       }
       if(PIXIT_SECC_CMN_CmValidate == cmValidate) {
           verdict := f_SECC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, fail);
       }
       if(verdict == pass) {
           verdict := f_SECC_CMN_TB_VTB_CmSlacMatch_001(fail);
       }
       if(verdict == pass) {
           tc_TT_match_join.start(par_TT_match_join);
           verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(false);
       }
       if(verdict == pass) {
           verdict := f_SECC_CMN_TB_VTB_PLCLinkStatus_001(fail);
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-----------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_AC_VTB_CmSlacParm_002() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions----------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   vc_activateNominal := false;
   preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

   //-------------- Test behavior-----------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_AC_TB_VTB_CmSlacParm_002(v_HAL_61851_Listener);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-----------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_AC_VTB_CmSlacParm_003() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;
   var verdicttype verdict;

   // -------------- Pre Conditions----------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   vc_testCaseSpecific := true;
   preConVerdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);

   //-------------- Test behavior-----------------------------------------------------
   if( preConVerdict == pass ) {
       verdict := f_SECC_AC_TB_VTB_CmSlacParm_001(v_HAL_61851_Listener);
       if(verdict == pass) {
           verdict := f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
       }
       if(verdict == pass) {
           verdict := f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
```

```
            }
            if(PIXIT_SECC_CMN_CmValidate == cmValidate) {
                verdict := f_SECC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, fail);
            }
            if(verdict == pass) {
                verdict := f_SECC_CMN_TB_VTB_CmSlacMatch_001(fail);
            }
            if(verdict == pass) {
                tc_TT_match_join.start(par_TT_match_join);
                verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(false);
            }
            if(verdict == pass) {
                verdict := f_SECC_CMN_TB_VTB_PLCLinkStatus_001(fail);
            }
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

     testcase TC_SECC_AC_VTB_CmSlacParm_004() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;
        var verdicttype verdict;

        // -------------- Pre Conditions--------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        vc_testCaseSpecific := true;
        preConVerdict := f_SECC_CMN_PR_SetProximityPilot_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            verdict := f_SECC_AC_TB_VTB_CmSlacParm_003(v_HAL_61851_Listener);
            if(verdict == pass) {
                verdict := f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
            }
            if(verdict == pass) {
                verdict := f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
            }
            if(PIXIT_SECC_CMN_CmValidate == cmValidate) {
                verdict := f_SECC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, fail);
            }
            if(verdict == pass) {
                verdict := f_SECC_CMN_TB_VTB_CmSlacMatch_001(fail);
            }
            if(verdict == pass) {
                tc_TT_match_join.start(par_TT_match_join);
                verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(false);
            }
            if(verdict == pass) {
                verdict := f_SECC_CMN_TB_VTB_PLCLinkStatus_001(fail);
            }
        } else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }
}
```

C.1.2 SECC test cases for AttenuationCharacterization

```
module TestCases_SECC_AttenuationCharacterization  {

    import from DataStructure_SLAC all;
    import from TestBehavior_SECC_AttenuationCharacterization all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_SECC_15118_3 all;
    import from PostConditions_SECC_15118_3 all;
    import from Templates_CMN_CmAttenCharRsp all;
import from Templates_CMN_SlacPayloadHeader all;
import from Templates_CMN_CmStartAttenCharInd all;
import from LibFunctions_15118_3  { group generalFunctions; }
import from TestBehavior_SECC_CmSlacParm all;

testcase TC_SECC_CMN_VTB_AttenuationCharacterization_001() runs on SECC_Tester
                                                           system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

   //-------------- Test behavior-----------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-----------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

 testcase TC_SECC_CMN_VTB_AttenuationCharacterization_002() runs on SECC_Tester
                                                            system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

    //-------------- Test behavior-----------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_AttenuationCharacterization_002();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-----------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_SECC_CMN_VTB_AttenuationCharacterization_003() runs on SECC_Tester
                                                            system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

    //-------------- Test behavior-----------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_AttenuationCharacterization_003(1);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-----------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_SECC_CMN_VTB_AttenuationCharacterization_004() runs on SECC_Tester
                                                            system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
```

```
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

        //-------------- Test behavior----------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_AttenuationCharacterization_004();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions---------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_SECC_CMN_VTB_AttenuationCharacterization_005() runs on SECC_Tester
                                                              system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

        //-------------- Test behavior----------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(
                                              m_CMN_CMN_SlacPayloadHeaderInvalid_001(),
                                              md_CMN_CMN_Acvarfield_001(
                                              par_testSystem_mac, vc_RunID));
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions---------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_SECC_CMN_VTB_AttenuationCharacterization_006() runs on SECC_Tester
                                                              system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

        //-------------- Test behavior----------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(
                                              m_CMN_CMN_SlacPayloadHeaderInvalid_002(),
                                              md_CMN_CMN_Acvarfield_001(
                                              par_testSystem_mac, vc_RunID));
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions---------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_SECC_CMN_VTB_AttenuationCharacterization_007() runs on SECC_Tester
                                                              system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

        //-------------- Test behavior----------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(
                                              m_CMN_CMN_SlacPayloadHeader_001(),
                                               md_CMN_CMN_Acvarfield_002(
                                               '000000000000'H, vc_RunID,
                                               '00000000000000000000000000000000'H,
                                               '00000000000000000000000000000000'H,
                                               '00'H));
     } else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions------------------------------------------------------
     f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

  testcase TC_SECC_CMN_VTB_AttenuationCharacterization_008() runs on SECC_Tester
                                                              system SystemSECC {

     var HAL_61851_Listener v_HAL_61851_Listener;
     var verdicttype preConVerdict;

     // -------------- Pre Conditions-----------------------------------------------------
     f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

     //-------------- Test behavior-------------------------------------------------------
     if( preConVerdict == pass ) {
         var RunID_TYPE v_RunID := f_randomHexStringGen(16);
         if(v_RunID != vc_RunID) {

             f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(
                                                 m_CMN_CMN_SlacPayloadHeader_001(),
                                                 md_CMN_CMN_Acvarfield_002(
                                                 par_testSystem_mac, v_RunID,
                                                 '00000000000000000000000000000000'H,
                                                 '00000000000000000000000000000000'H,
                                                 '00'H));

         } else {setverdict(inconc, "Invalid runID is equal to current runID.")};
     } else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions------------------------------------------------------
     f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

  testcase TC_SECC_CMN_VTB_AttenuationCharacterization_009() runs on SECC_Tester
                                                              system SystemSECC {

     var HAL_61851_Listener v_HAL_61851_Listener;
     var verdicttype preConVerdict;

     // -------------- Pre Conditions-----------------------------------------------------
     f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

     //-------------- Test behavior-------------------------------------------------------
     if( preConVerdict == pass ) {
         f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(
                                               m_CMN_CMN_SlacPayloadHeader_001(),
                                               md_CMN_CMN_Acvarfield_002(
                                               par_testSystem_mac, vc_RunID,
                                               '00000000000000000000000000000001'H,
                                               '00000000000000000000000000000000'H,
                                               '00'H));
     } else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions------------------------------------------------------
     f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

  testcase TC_SECC_CMN_VTB_AttenuationCharacterization_010() runs on SECC_Tester
                                                              system SystemSECC {
```

```
   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(
                                               m_CMN_CMN_SlacPayloadHeader_001(),
                                               md_CMN_CMN_Acvarfield_002(
                                               par_testSystem_mac, vc_RunID,
                                               '00000000000000000000000000000000'H,
                                               '00000000000000000000000000000001'H,
                                               '00'H));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_AttenuationCharacterization_011() runs on SECC_Tester
                                                            system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(
                                               m_CMN_CMN_SlacPayloadHeader_001(),
                                               md_CMN_CMN_Acvarfield_002(
                                               par_testSystem_mac, vc_RunID,
                                               '00000000000000000000000000000000'H,
                                               '00000000000000000000000000000000'H,
                                               'FF'H));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_AttenuationCharacterization_012() runs on SECC_Tester
                                                            system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_AttenuationCharacterization_006();
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_AttenuationCharacterization_013() runs on SECC_Tester
                                                            system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

   //-------------- Test behavior-------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_AttenuationCharacterization_007(
                                        md_CMN_CMN_CmStartAttenCharInd_001(
                                        m_CMN_CMN_SlacPayloadHeaderInvalid_001(),
                                        vc_Num_sounds, vc_Time_out, '01'H,
                                        par_testSystem_mac, vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_AttenuationCharacterization_014() runs on SECC_Tester
                                                            system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

   //-------------- Test behavior-------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_AttenuationCharacterization_007(
                                        md_CMN_CMN_CmStartAttenCharInd_001(
                                        m_CMN_CMN_SlacPayloadHeaderInvalid_002(),
                                        vc_Num_sounds, vc_Time_out, '01'H,
                                        par_testSystem_mac, vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_AttenuationCharacterization_015() runs on SECC_Tester
                                                            system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

   //-------------- Test behavior-------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_AttenuationCharacterization_007(
                                        md_CMN_CMN_CmStartAttenCharInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        '00'H, vc_Time_out, '01'H,
                                        par_testSystem_mac, vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_AttenuationCharacterization_016() runs on SECC_Tester
                                                            system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
```

```
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_AttenuationCharacterization_007(
                                        md_CMN_CMN_CmStartAttenCharInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        vc_Num_sounds, '00'H, '01'H,
                                        par_testSystem_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_SECC_CMN_VTB_AttenuationCharacterization_017() runs on SECC_Tester
                                                            system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_AttenuationCharacterization_007(
                                        md_CMN_CMN_CmStartAttenCharInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        vc_Num_sounds, vc_Time_out, '00'H,
                                        par_testSystem_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_SECC_CMN_VTB_AttenuationCharacterization_018() runs on SECC_Tester
                                                            system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        var RunID_TYPE v_RunID := f_randomHexStringGen(16);
        if(v_RunID != vc_RunID) {

            f_SECC_CMN_TB_VTB_AttenuationCharacterization_007(
                                              md_CMN_CMN_CmStartAttenCharInd_001(
                                              m_CMN_CMN_SlacPayloadHeader_001(),
                                              vc_Num_sounds, vc_Time_out, '01'H,
                                              par_testSystem_mac, v_RunID));

        } else {setverdict(inconc, "Invalid runID is equal to current runID.")};
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }
```

```
    testcase TC_SECC_CMN_VTB_AttenuationCharacterization_019() runs on SECC_Tester
                                                                system SystemSECC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-------------------------------------------------------
       f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

       //-------------- Test behavior--------------------------------------------------------
       if( preConVerdict == pass ) {
           f_SECC_CMN_TB_VTB_AttenuationCharacterization_008(v_HAL_61851_Listener);
       } else {
           log("PreCondition was unsuccessful.");
       }

       //------------- Post Conditions-------------------------------------------------------
       f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

    testcase TC_SECC_CMN_VTB_AttenuationCharacterization_020() runs on SECC_Tester
                                                               system SystemSECC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;
       var verdicttype verdict;
       var AttenProfile_TYPE v_attenuation_list;

       // -------------- Pre Conditions-------------------------------------------------------
       f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);

       //-------------- Test behavior--------------------------------------------------------
       if( preConVerdict == pass ) {
           f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
           if(getverdict == pass) {
               v_attenuation_list := vc_attenuation_list;
               f_SECC_CMN_Reset_001(v_HAL_61851_Listener);
           }
           if(getverdict == pass) {
               f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
           }
           if(getverdict == pass) {
               f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
           }
           if(getverdict == pass) {
               f_SECC_CMN_compareAttenuationValues_001(v_attenuation_list,
                                                       vc_attenuation_list);
           }
       } else {
           log("PreCondition was unsuccessful.");
       }


       //------------- Post Conditions-------------------------------------------------------
       f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }
}
```

C.1.3 SECC test cases for CmValidate

```
module TestCases_SECC_CmValidate {

    import from TestBehavior_SECC_CmValidate all;
    import from TestBehavior_SECC_CmSlacMatch all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_SECC_15118_3 all;
    import from PostConditions_SECC_15118_3 all;
    import from Templates_CMN_CmValidate all;
    import from Timer_15118_3 all;

    testcase TC_SECC_CMN_VTB_CmValidate_001() runs on SECC_Tester system SystemSECC {
        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions----------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

        //-------------- Test behavior-----------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, fail);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions----------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

      testcase TC_SECC_CMN_VTB_CmValidate_002() runs on SECC_Tester system SystemSECC {

         var HAL_61851_Listener v_HAL_61851_Listener;
         var verdicttype preConVerdict;

         // -------------- Pre Conditions----------------------------------------------------------
         f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
         preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

         //-------------- Test behavior-----------------------------------------------------------
         if( preConVerdict == pass ) {
             f_SECC_CMN_TB_VTB_CmValidate_002();
         } else {
             log("PreCondition was unsuccessful.");
         }

         //------------- Post Conditions----------------------------------------------------------
         f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
         f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       }

      testcase TC_SECC_CMN_VTB_CmValidate_003() runs on SECC_Tester system SystemSECC {

         var HAL_61851_Listener v_HAL_61851_Listener;
         var verdicttype preConVerdict;

         // -------------- Pre Conditions----------------------------------------------------------
         f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
         preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

         //-------------- Test behavior-----------------------------------------------------------
         if( preConVerdict == pass ) {
             f_SECC_CMN_TB_VTB_CmValidate_003(v_HAL_61851_Listener);
         } else {
             log("PreCondition was unsuccessful.");
         }

         //------------- Post Conditions----------------------------------------------------------
         f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
         f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       }

      testcase TC_SECC_CMN_VTB_CmValidate_004() runs on SECC_Tester system SystemSECC {

         var HAL_61851_Listener v_HAL_61851_Listener;
         var verdicttype preConVerdict;

         // -------------- Pre Conditions----------------------------------------------------------
         f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
         preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

         //-------------- Test behavior-----------------------------------------------------------
         if( preConVerdict == pass ) {
             var hexstring v_pilotTimer := int2hex(float2int((par_TP_EV_vald_toggle * 10.0) - 1.0),2);
             f_SECC_CMN_TB_VTB_CmValidate_004(v_HAL_61851_Listener,
                                              md_CMN_CMN_CmValidateReq_004(
                                              'FF'H, v_pilotTimer, '01'H));
         } else {
             log("PreCondition was unsuccessful.");
         }
```

```
   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmValidate_005() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       var hexstring v_pilotTimer := int2hex(float2int((par_TP_EV_vald_toggle * 10.0) - 1.0),2);
       f_SECC_CMN_TB_VTB_CmValidate_005(md_CMN_CMN_CmValidateReq_004(
                                        '00'H, v_pilotTimer,
                                        par_cmValidate_result_notReady));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmValidate_006() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       var hexstring v_pilotTimer := int2hex(float2int((par_TP_EV_vald_toggle * 10.0) - 1.0),2);
       f_SECC_CMN_TB_VTB_CmValidate_005(md_CMN_CMN_CmValidateReq_004(
                                        '00'H, v_pilotTimer,
                                        par_cmValidate_result_success));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmValidate_007() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       var hexstring v_pilotTimer := int2hex(float2int((par_TP_EV_vald_toggle * 10.0) - 1.0),2);
       f_SECC_CMN_TB_VTB_CmValidate_005(md_CMN_CMN_CmValidateReq_004(
                                        '00'H, v_pilotTimer,
                                        par_cmValidate_result_failure));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }
```

```
testcase TC_SECC_CMN_VTB_CmValidate_008() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior--------------------------------------------------------
   if( preConVerdict == pass ) {
       var hexstring v_pilotTimer := int2hex(float2int((par_TP_EV_vald_toggle * 10.0) - 1.0),2);
       f_SECC_CMN_TB_VTB_CmValidate_005(md_CMN_CMN_CmValidateReq_004(
                                        '00'H, v_pilotTimer,
                                        par_cmValidate_result_notRequired));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmValidate_009() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var SLAC_Tester2 v_SLAC_Tester2;
   var SLAC_Tester3 v_SLAC_Tester3;
   var SLAC_Tester4 v_SLAC_Tester4;
   var SLAC_Tester5 v_SLAC_Tester5;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2,
                                            v_SLAC_Tester3, v_SLAC_Tester4,
                                            v_SLAC_Tester5, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);
   if( preConVerdict == pass ) {
       tc_TP_EVSE_match_session.start(par_TT_EVSE_match_session);
       v_SLAC_Tester2.start(f_SECC_CMN_TB_VTB_CmValidatePreCondition_001());
       v_SLAC_Tester2.done;
   }

   //-------------- Test behavior--------------------------------------------------------
   if( getverdict == pass ) {
       f_SECC_CMN_TB_VTB_CmValidate_006();
       if(getverdict == pass) {
           v_SLAC_Tester2.start(f_SECC_CMN_TB_VTB_CmValidate_009());
           f_SECC_CMN_TB_VTB_CmValidate_007(v_HAL_61851_Listener);
           v_SLAC_Tester2.done;
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2,
                                                v_SLAC_Tester3, v_SLAC_Tester4,
                                                v_SLAC_Tester5, system);
}

testcase TC_SECC_CMN_VTB_CmValidate_010() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior--------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_CmValidate_008(par_cmValidate_result_failure);
       if(getverdict == pass) {
           setverdict(pass,"CM_VALIDATE.CNF message with 'failure' is correct. " &
                           "SUT is not able to perform any BCB-Toggle.");
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmValidate_011() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_CmValidate_008(par_cmValidate_result_notRequired);
       setverdict(pass,"CM_VALIDATE.CNF message with 'notRequired' is correct. " &
                       "SUT has indicated that a validation is not required.");
       if(getverdict == pass) {
           f_SECC_CMN_TB_VTB_CmValidate_007(v_HAL_61851_Listener);
           if(getverdict == pass) {
               setverdict(pass,"SUT has finished the validation process.");
           }
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmValidate_012() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_CmValidate_008(par_cmValidate_result_notRequired);
       setverdict(pass,"CM_VALIDATE.CNF message with 'notRequired' is correct. " &
                       "SUT has indicated that a validation is not required.");
       if(getverdict == pass) {
           f_SECC_CMN_TB_VTB_CmSlacMatch_001(fail);
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_CMN_VTB_CmValidate_013() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_CmValidate_010(v_HAL_61851_Listener);
   } else {
       log("PreCondition was unsuccessful.");
```

```
        }


        //------------- Post Conditions-------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }
}
```

C.1.4 SECC test cases for CmSlacMatch

```
module TestCases_SECC_CmSlacMatch {

    import from DataStructure_SLAC all;
    import from TestBehavior_SECC_CmSlacMatch all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_SECC_15118_3 all;
    import from PostConditions_SECC_15118_3 all;
    import from Templates_CMN_CmSlacMatch all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from LibFunctions_15118_3  { group generalFunctions; }

    testcase TC_SECC_CMN_VTB_CmSlacMatch_001() runs on SECC_Tester system SystemSECC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-------------------------------------------------------
       f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

       //-------------- Test behavior-------------------------------------------------------
       if( preConVerdict == pass ) {
           f_SECC_CMN_TB_VTB_CmSlacMatch_001(fail);
       } else {
           log("PreCondition was unsuccessful.");
       }

       //------------- Post Conditions-------------------------------------------------------
       f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

     testcase TC_SECC_CMN_VTB_CmSlacMatch_002() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

        //-------------- Test behavior-------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmSlacMatch_001(fail);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_SECC_CMN_VTB_CmSlacMatch_003() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

        //-------------- Test behavior-------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmSlacMatch_002();
        } else {
```

```
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmSlacMatch_004() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

    //-------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_002();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmSlacMatch_005() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

    //-------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_003();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }


testcase TC_SECC_CMN_VTB_CmSlacMatch_006() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

    //-------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_003();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmSlacMatch_007() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                      m_CMN_CMN_SlacPayloadHeaderInvalid_001(), '003E'H,
                                      '00000000000000000000000000000000'H,
                                      par_testSystem_mac,
                                      '00000000000000000000000000000000'H,
                                      vc_sut_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_SECC_CMN_VTB_CmSlacMatch_008() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                      m_CMN_CMN_SlacPayloadHeaderInvalid_001(), '003E'H,
                                      '00000000000000000000000000000000'H,
                                      par_testSystem_mac,
                                      '00000000000000000000000000000000'H,
                                      vc_sut_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_SECC_CMN_VTB_CmSlacMatch_009() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                      m_CMN_CMN_SlacPayloadHeaderInvalid_002(), '003E'H,
                                      '00000000000000000000000000000000'H,
                                      par_testSystem_mac,
                                      '00000000000000000000000000000000'H,
                                      vc_sut_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_SECC_CMN_VTB_CmSlacMatch_010() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

    //-------------- Test behavior----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                          m_CMN_CMN_SlacPayloadHeaderInvalid_002(), '003E'H,
                                          '00000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          '00000000000000000000000000000000'H,
                                          vc_sut_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions----------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmSlacMatch_011() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions---------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

    //-------------- Test behavior----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                          m_CMN_CMN_SlacPayloadHeader_001(), '0000'H,
                                          '00000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          '00000000000000000000000000000000'H,
                                          vc_sut_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions----------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmSlacMatch_012() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions---------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

    //-------------- Test behavior----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                          m_CMN_CMN_SlacPayloadHeader_001(), '0000'H,
                                          '00000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          '00000000000000000000000000000000'H,
                                          vc_sut_mac, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions----------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_SECC_CMN_VTB_CmSlacMatch_013() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions---------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
```

```
      preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

      //-------------- Test behavior----------------------------------------------------------
      if( preConVerdict == pass ) {
          f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                            m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                            '00000000000000000000000000000001'H,
                                            par_testSystem_mac,
                                            '00000000000000000000000000000000'H,
                                            vc_sut_mac, vc_RunID));
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions---------------------------------------------------------
      f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

   testcase TC_SECC_CMN_VTB_CmSlacMatch_014() runs on SECC_Tester system SystemSECC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions--------------------------------------------------------
      f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

      //-------------- Test behavior----------------------------------------------------------
      if( preConVerdict == pass ) {
          f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                            m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                            '00000000000000000000000000000001'H,
                                            par_testSystem_mac,
                                            '00000000000000000000000000000000'H,
                                            vc_sut_mac, vc_RunID));
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions---------------------------------------------------------
      f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

   testcase TC_SECC_CMN_VTB_CmSlacMatch_015() runs on SECC_Tester system SystemSECC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions--------------------------------------------------------
      f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

      //-------------- Test behavior----------------------------------------------------------
      if( preConVerdict == pass ) {
          f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                            m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                            '00000000000000000000000000000000'H,
                                            '000000000000'H,
                                            '00000000000000000000000000000000'H,
                                            vc_sut_mac, vc_RunID));
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions---------------------------------------------------------
      f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

   testcase TC_SECC_CMN_VTB_CmSlacMatch_016() runs on SECC_Tester system SystemSECC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions--------------------------------------------------------
      f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);
   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                         m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                         '0000000000000000000000000000000000'H,
                                         '000000000000'H,
                                         '0000000000000000000000000000000000'H,
                                         vc_sut_mac, vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions----------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmSlacMatch_017() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions---------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                         m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                         '0000000000000000000000000000000000'H,
                                         par_testSystem_mac,
                                         '0000000000000000000000000000000001'H,
                                         vc_sut_mac, vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions----------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmSlacMatch_018() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions---------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                         m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                         '0000000000000000000000000000000000'H,
                                         par_testSystem_mac,
                                         '0000000000000000000000000000000001'H,
                                         vc_sut_mac, vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions----------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmSlacMatch_019() runs on SECC_Tester system SystemSECC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions---------------------------------------------------------
   f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);
```

```
    //-------------- Test behavior-----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                          m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                          '0000000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          '0000000000000000000000000000000000'H,
                                          '000000000000'H, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-----------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmSlacMatch_020() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions---------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

    //-------------- Test behavior-----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                          m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                          '0000000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          '0000000000000000000000000000000000'H,
                                          '000000000000'H, vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-----------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmSlacMatch_021() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions---------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

    //-------------- Test behavior-----------------------------------------------------------
    if( preConVerdict == pass ) {
        var RunID_TYPE v_RunID := f_randomHexStringGen(16);
        if(v_RunID != vc_RunID) {

            f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                              m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                              '0000000000000000000000000000000000'H,
                                              par_testSystem_mac,
                                              '0000000000000000000000000000000000'H,
                                              vc_sut_mac, v_RunID));

        } else {setverdict(inconc, "Invalid runID is equal to current runID.")};
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-----------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmSlacMatch_022() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;
```

```
        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            var RunID_TYPE v_RunID := f_randomHexStringGen(16);
            if(v_RunID != vc_RunID) {

                f_SECC_CMN_TB_VTB_CmSlacMatch_004(md_CMN_CMN_CmSlacMatchReq_002(
                                                  m_CMN_CMN_SlacPayloadHeader_001(), '003E'H,
                                                  '0000000000000000000000000000000000'H,
                                                  par_testSystem_mac,
                                                  '0000000000000000000000000000000000'H,
                                                  vc_sut_mac, v_RunID));

            } else {setverdict(inconc, "Invalid runID is equal to current runID.")};
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_SECC_CMN_VTB_CmSlacMatch_023() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmSlacMatch_005(v_HAL_61851_Listener);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_SECC_CMN_VTB_CmSlacMatch_024() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmSlacMatch_005(v_HAL_61851_Listener);
        } else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions-------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }
}
```

C.1.5 SECC test cases for PLCLinkStatus

```
module TestCases_SECC_CmAmpMap {

    import from DataStructure_SLAC all;
    import from TestBehavior_SECC_CmAmpMap all;
    import from TestBehavior_SECC_PLCLinkStatus all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_SECC_15118_3 all;
    import from Timer_15118_3 all;
    import from PostConditions_SECC_15118_3 all;
    import from Services_TXPowerLimitation all;

   testcase TC_SECC_CMN_VTB_CmAmpMap_001() runs on SECC_Tester system SystemSECC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions--------------------------------------------------------
       f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

       //-------------- Test behavior---------------------------------------------------------
       if( preConVerdict == pass ) {
           f_SECC_CMN_TB_VTB_CmAmpMap_001(fail);
           if(getverdict == pass) {
              f_SECC_CMN_checkTXPowerLimitation();
           }
       } else {
           log("PreCondition was unsuccessful.");
       }

       //------------- Post Conditions--------------------------------------------------------
       f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

   testcase TC_SECC_CMN_VTB_CmAmpMap_002() runs on SECC_Tester system SystemSECC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions--------------------------------------------------------
       f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

       //-------------- Test behavior---------------------------------------------------------
       if( preConVerdict == pass ) {
           f_SECC_CMN_TB_VTB_CmAmpMap_002(fail);
       } else {
           log("PreCondition was unsuccessful.");
       }

       //------------- Post Conditions--------------------------------------------------------
       f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_SECC_CMN_VTB_CmAmpMap_003() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions--------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_003();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_SECC_CMN_VTB_CmAmpMap_004() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;
        // -------------- Pre Conditions-----------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_004();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-----------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

    testcase TC_SECC_CMN_VTB_CmAmpMap_005() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-----------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_005();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-----------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_SECC_CMN_VTB_CmAmpMap_006() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-----------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_006();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-----------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_SECC_CMN_VTB_CmAmpMap_007() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-----------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_007();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-----------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }
    testcase TC_SECC_CMN_VTB_CmAmpMap_008() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-----------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmAmpMap_001(v_HAL_61851_Listener);

        //-------------- Test behavior-------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_008();
        } else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions-----------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }
}
```

C.1.6 SECC test cases for CmAmpMap

```
module TestCases_SECC_CmAmpMap {

    import from DataStructure_SLAC all;
    import from TestBehavior_SECC_CmAmpMap all;
    import from TestBehavior_SECC_PLCLinkStatus all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_SECC_15118_3 all;
    import from Timer_15118_3 all;
    import from PostConditions_SECC_15118_3 all;
    import from Services_TXPowerLimitation all;

   testcase TC_SECC_CMN_VTB_CmAmpMap_001() runs on SECC_Tester system SystemSECC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-----------------------------------------------------
       f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

       //-------------- Test behavior-------------------------------------------------------
       if( preConVerdict == pass ) {
           f_SECC_CMN_TB_VTB_CmAmpMap_001(fail);
           if(getverdict == pass) {
              f_SECC_CMN_checkTXPowerLimitation();
           }
       } else {
           log("PreCondition was unsuccessful.");
       }

       //------------- Post Conditions-----------------------------------------------------
       f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

   testcase TC_SECC_CMN_VTB_CmAmpMap_002() runs on SECC_Tester system SystemSECC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-----------------------------------------------------
       f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

       //-------------- Test behavior-------------------------------------------------------
       if( preConVerdict == pass ) {
           f_SECC_CMN_TB_VTB_CmAmpMap_002(fail);
       } else {
           log("PreCondition was unsuccessful.");
       }
   //------------- Post Conditions-----------------------------------------------------------
   f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_CMN_VTB_CmAmpMap_003() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmAmpMap_003();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_CMN_VTB_CmAmpMap_004() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmAmpMap_004();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_SECC_CMN_VTB_CmAmpMap_005() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior----------------------------------------------------------
    if( preConVerdict == pass ) {
        f_SECC_CMN_TB_VTB_CmAmpMap_005();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_SECC_CMN_VTB_CmAmpMap_006() runs on SECC_Tester system SystemSECC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior----------------------------------------------------------
```

```
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_006();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_SECC_CMN_VTB_CmAmpMap_007() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior--------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_007();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_SECC_CMN_VTB_CmAmpMap_008() runs on SECC_Tester system SystemSECC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_SECC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_SECC_CMN_PR_CmAmpMap_001(v_HAL_61851_Listener);

        //-------------- Test behavior--------------------------------------------------------
        if( preConVerdict == pass ) {
            f_SECC_CMN_TB_VTB_CmAmpMap_008();
        } else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions--------------------------------------------------------
        f_SECC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }
}
```

C.2 EVCC + PLC bridge test cases

This subclause includes all test case *specifications* where the EV is defined as SUT.

C.2.1 EVCC test cases for CmSlacParm

```
module TestCases_EVCC_CmSlacParm {

    import from DataStructure_SLAC all;
    import from TestBehavior_EVCC_CmSlacParm all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from Timer_15118_3 all;
    import from PreConditions_EVCC_15118_3 all;
    import from PostConditions_EVCC_15118_3 all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmSlacParm all;
    import from LibFunctions_15118_3  { group generalFunctions; }

    testcase TC_EVCC_CMN_VTB_CmSlacParm_001() runs on EVCC_Tester system SystemEVCC {

       var HAL_61851_Listener v_HAL_61851_Listener;
```

```
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacParm_002() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_002();
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_EVCC_CMN_VTB_CmSlacParm_003() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_003(md_CMN_CMN_CmSlacParmCnf_002('000000000000'H,
                                           '0A'H, '06'H, '01'H, vc_sut_mac,
                                        m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

 testcase TC_EVCC_CMN_VTB_CmSlacParm_004() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_CmSlacParm_003(md_CMN_CMN_CmSlacParmCnf_002('FFFFFFFFFFFF'H,
                                         '0A'H, '00'H, '01'H, vc_sut_mac,
                                         m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID));
    } else {
        log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_EVCC_CMN_VTB_CmSlacParm_005() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_003(md_CMN_CMN_CmSlacParmCnf_002('FFFFFFFFFFFF'H,
                                      '0A'H, '06'H, '00'H, vc_sut_mac,
                                      m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_EVCC_CMN_VTB_CmSlacParm_006() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_003(md_CMN_CMN_CmSlacParmCnf_002('FFFFFFFFFFFF'H,
                                      '0A'H, '06'H, '01'H, vc_sut_mac,
                                      m_CMN_CMN_SlacPayloadHeaderInvalid_001(), vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_EVCC_CMN_VTB_CmSlacParm_007() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_003(md_CMN_CMN_CmSlacParmCnf_002('FFFFFFFFFFFF'H,
                                      '0A'H, '06'H, '01'H, vc_sut_mac,
                                      m_CMN_CMN_SlacPayloadHeaderInvalid_002(), vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

testcase TC_EVCC_CMN_VTB_CmSlacParm_008() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
```

```
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        var RunID_TYPE v_RunID := f_randomHexStringGen(16);
        if(v_RunID != vc_RunID) {

            f_EVCC_CMN_TB_VTB_CmSlacParm_003(md_CMN_CMN_CmSlacParmCnf_002('FFFFFFFFFFFF'H,
                                             '0A'H, '06'H, '01'H, vc_sut_mac,
                                             m_CMN_CMN_SlacPayloadHeader_001(), v_RunID));
        } else {setverdict(inconc, "Invalid runID is equal to current runID.")};
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

 testcase TC_EVCC_CMN_VTB_CmSlacParm_009() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_CmSlacParm_004();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_EVCC_CMN_VTB_CmSlacParm_010() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_CmSlacParm_005();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
  }

 testcase TC_EVCC_CMN_VTB_CmSlacParm_011() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions--------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_CmSlacParm_003(md_CMN_CMN_CmSlacParmCnf_002('FFFFFFFFFFFF'H,
                                           '0A'H, '06'H, '01'H, '000000000000'H,
                                           m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacParm_012() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   vc_errorState := e_NegVolt12;
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior-------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacParm_013() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   vc_DutyCycle := 100;
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior-------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacParm_014() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   vc_errorState := e_NegVolt12;
   vc_DutyCycle := 100;
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior-------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_AC_VTB_CmSlacParm_001() runs on EVCC_Tester system SystemEVCC {
        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        vc_DutyCycle := cc_dutyCycle_32A;
        preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

        //-------------- Test behavior--------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

     testcase TC_EVCC_AC_VTB_CmSlacParm_002() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        vc_DutyCycle := cc_dutyCycle_32A;
        vc_errorState := e_NegVolt12;
        preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

        //-------------- Test behavior--------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        } else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions-------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }
}
```

C.2.2 EVCC test cases for AttenuationCharacterization

```
module TestCases_EVCC_AttenuationCharacterization {

    import from DataStructure_SLAC all;
    import from PreConditions_EVCC_15118_3 all;
    import from TestBehavior_EVCC_AttenuationCharacterization all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PostConditions_EVCC_15118_3 all;
    import from Timer_15118_3 all;
    import from Templates_EVCC_CmAttenCharInd all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from TestBehavior_EVCC_CmSlacParm all;
    import from LibFunctions_15118_3  { group generalFunctions; }

    testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_001() runs on EVCC_Tester
                                                            system SystemEVCC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-------------------------------------------------------
       f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
       preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

       //-------------- Test behavior--------------------------------------------------------
       if( preConVerdict == pass ) {
           f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
       } else {
           log("PreCondition was unsuccessful.");
```

```
      }

      //------------- Post Conditions------------------------------------------------------
      f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
    }

   testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_002() runs on EVCC_Tester
                                                              system SystemEVCC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var SLAC_Tester2 v_SLAC_Tester2;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions-----------------------------------------------------
      f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
      preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

      //-------------- Test behavior-------------------------------------------------------
      if( preConVerdict == pass ) {
          v_SLAC_Tester2.start(f_EVCC_CMN_TB_VTB_AttenuationCharacterization_002());
          var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
          if(testbehaviorVerdict == pass) {
                  f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
          }
          v_SLAC_Tester2.done;
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions------------------------------------------------------
      f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
    }

   testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_003() runs on EVCC_Tester
                                                              system SystemEVCC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions-----------------------------------------------------
      f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
      preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

      //-------------- Test behavior-------------------------------------------------------
      if( preConVerdict == pass ) {
          f_EVCC_CMN_TB_VTB_AttenuationCharacterization_004();
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions------------------------------------------------------
      f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
    }

    testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_004() runs on EVCC_Tester
                                                                system SystemEVCC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-----------------------------------------------------
       f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
       preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

       //-------------- Test behavior-------------------------------------------------------
       if( preConVerdict == pass ) {
           f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                    m_CMN_CMN_SlacPayloadHeaderInvalid_001(),
                                                    vc_sut_mac, vc_RunID, vc_Num_sounds,
                                                    '00000000000000000000000000000000'H,
                                                    '00000000000000000000000000000000'H,
                                                    '3A'H, m_EVCC_CMN_atten_list_001()));
       } else {
           log("PreCondition was unsuccessful.");
       }
   //------------- Post Conditions--------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
 }

testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_005() runs on EVCC_Tester
                                                            system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
   preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                 m_CMN_CMN_SlacPayloadHeaderInvalid_002(),
                                                 vc_sut_mac, vc_RunID, vc_Num_sounds,
                                                 '0000000000000000000000000000000000'H,
                                                 '0000000000000000000000000000000000'H,
                                                 '3A'H, m_EVCC_CMN_atten_list_001()));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
 }

testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_006() runs on EVCC_Tester
                                                            system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
   preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                 m_CMN_CMN_SlacPayloadHeader_001(),
                                                 '000000000000'H, vc_RunID,
                                                 vc_Num_sounds,
                                                 '0000000000000000000000000000000000'H,
                                                 '0000000000000000000000000000000000'H,
                                                 '3A'H, m_EVCC_CMN_atten_list_001()));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
 }

testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_007() runs on EVCC_Tester
                                                            system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
   preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       var RunID_TYPE v_RunID := f_randomHexStringGen(16);
       if(v_RunID != vc_RunID) {

           f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                 m_CMN_CMN_SlacPayloadHeader_001(),
                                                 vc_sut_mac, v_RunID, vc_Num_sounds,
                                                     '00000000000000000000000000000000'H,
                                                     '00000000000000000000000000000000'H,
                                                     '3A'H, m_EVCC_CMN_atten_list_001()));

         } else {setverdict(inconc, "Invalid runID is equal to current runID.")};
     } else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions----------------------------------------------------
     f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
   }

  testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_008() runs on EVCC_Tester
                                                            system SystemEVCC {

     var HAL_61851_Listener v_HAL_61851_Listener;
     var verdicttype preConVerdict;

     // -------------- Pre Conditions---------------------------------------------------
     f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
     preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

     //-------------- Test behavior----------------------------------------------------
     if( preConVerdict == pass ) {
         f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                     m_CMN_CMN_SlacPayloadHeader_001(),
                                                     vc_sut_mac, vc_RunID, vc_Num_sounds,
                                                     '00000000000000000000000000000001'H,
                                                     '00000000000000000000000000000000'H,
                                                     '3A'H, m_EVCC_CMN_atten_list_001()));
     } else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions----------------------------------------------------
     f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
   }

  testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_009() runs on EVCC_Tester
                                                            system SystemEVCC {

     var HAL_61851_Listener v_HAL_61851_Listener;
     var verdicttype preConVerdict;

     // -------------- Pre Conditions---------------------------------------------------
     f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
     preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

     //-------------- Test behavior----------------------------------------------------
     if( preConVerdict == pass ) {
         f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                     m_CMN_CMN_SlacPayloadHeader_001(),
                                                     vc_sut_mac, vc_RunID, vc_Num_sounds,
                                                     '00000000000000000000000000000000'H,
                                                     '00000000000000000000000000000001'H,
                                                     '3A'H, m_EVCC_CMN_atten_list_001()));
     } else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions----------------------------------------------------
     f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
   }

  testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_010() runs on EVCC_Tester
                                                            system SystemEVCC {

     var HAL_61851_Listener v_HAL_61851_Listener;
     var verdicttype preConVerdict;

     // -------------- Pre Conditions---------------------------------------------------
     f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
     preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

     //-------------- Test behavior----------------------------------------------------
```

```
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                  m_CMN_CMN_SlacPayloadHeader_001(),
                                                  vc_sut_mac, vc_RunID, vc_Num_sounds,
                                                  '00000000000000000000000000000000'H,
                                                  '00000000000000000000000000000000'H,
                                                  '00'H, m_EVCC_CMN_atten_list_001()));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------ Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
  }

 testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_011() runs on EVCC_Tester
                                                            system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // ------------- Pre Conditions------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
    preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

    //------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(md_EVCC_CMN_CmAttenCharInd_002(
                                                  m_CMN_CMN_SlacPayloadHeader_001(),
                                                  vc_sut_mac, vc_RunID, '00'H,
                                                  '00000000000000000000000000000000'H,
                                                  '00000000000000000000000000000000'H,
                                                  '3A'H, m_EVCC_CMN_atten_list_001()));
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------ Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
  }

 testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_012() runs on EVCC_Tester
                                                            system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // ------------- Pre Conditions------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
    preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

    //------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_AttenuationCharacterization_006(e_OscOff);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------ Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
  }

 testcase TC_EVCC_CMN_VTB_AttenuationCharacterization_013() runs on EVCC_Tester
                                                            system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // ------------- Pre Conditions------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
    preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

    //------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_AttenuationCharacterization_006(e_NegVolt12);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
      }

     testcase TC_EVCC_AC_VTB_AttenuationCharacterization_001() runs on EVCC_Tester
                                                                system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
        preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_AC_TB_VTB_AttenuationCharacterization_001(10);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
      }

     testcase TC_EVCC_AC_VTB_AttenuationCharacterization_002() runs on EVCC_Tester
                                                                system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener,system);
        preConVerdict := f_EVCC_CMN_PR_CmSlacParm_001 (v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_AC_TB_VTB_AttenuationCharacterization_001(96);
        } else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener,system);
      }
}
```

C.2.3 EVCC test cases for CmValidate

```
module TestCases_EVCC_CmValidate {

    import from TestBehavior_EVCC_CmValidate all;
    import from TestBehavior_EVCC_CmSlacParm all;
    import from TestBehavior_EVCC_CmSlacMatch all;
    import from TestBehavior_EVCC_AttenuationCharacterization all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_EVCC_15118_3 all;
    import from PostConditions_EVCC_15118_3 all;
    import from Templates_CMN_CmValidate all;
    import from Timer_15118_3 all;

     testcase TC_EVCC_CMN_VTB_CmValidate_001() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);
```

```
   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, false,
                                        vc_DutyCycle, fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_002() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, true,
                                        10, fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_003() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, true,
                                        96, fail);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_004() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmValidate_002();
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}
```

```
testcase TC_EVCC_CMN_VTB_CmValidate_005() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmValidate_003(md_CMN_CMN_CmValidateCnf_003('FF'H, '00'H, '01'H));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_006() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmValidate_003(md_CMN_CMN_CmValidateCnf_003('00'H, 'FF'H, '01'H));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_007() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var SLAC_Tester2 v_SLAC_Tester2;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------------
   if( preConVerdict == pass ) {
       v_SLAC_Tester2.start(f_EVCC_CMN_TB_VTB_CmValidate_009());
       var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
       if(testbehaviorVerdict == pass) {
           testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
       }
       if(testbehaviorVerdict == pass) {
           f_EVCC_CMN_TB_VTB_CmValidate_002();
       }
       v_SLAC_Tester2.done;
       if(getverdict == pass) {
           setverdict(pass, "SUT has continued the SLAC validation process " &
                            "with the next potential EVSE");
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_008() runs on EVCC_Tester system SystemEVCC {
   var HAL_61851_Listener v_HAL_61851_Listener;
   var SLAC_Tester2 v_SLAC_Tester2;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior------------------------------------------------------
   if( preConVerdict == pass ) {
       v_SLAC_Tester2.start(f_EVCC_CMN_TB_VTB_CmValidate_010(par_cmValidate_result_success));
       var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
       if(testbehaviorVerdict == pass) {
           testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
       }
       if(testbehaviorVerdict == pass) {
           f_EVCC_CMN_TB_VTB_CmValidate_004(par_cmValidate_result_success, true);
       }
       v_SLAC_Tester2.done;
       if(getverdict == pass) {
           setverdict(pass, "SUT has continued the SLAC validation process " &
                            "with the next potential EVSE");
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-----------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_009() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var SLAC_Tester2 v_SLAC_Tester2;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior------------------------------------------------------
   if( preConVerdict == pass ) {
       v_SLAC_Tester2.start(f_EVCC_CMN_TB_VTB_CmValidate_011(v_HAL_61851_Listener));
       var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
       if(testbehaviorVerdict == pass) {
           testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
       }
       if(testbehaviorVerdict == pass) {
           f_EVCC_CMN_TB_VTB_CmValidate_005(v_HAL_61851_Listener);
       }
       v_SLAC_Tester2.done;
       if(getverdict == pass) {
           setverdict(pass, "SUT has continued the SLAC validation process " &
                            "with the next potential EVSE");
       }
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-----------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_010() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var SLAC_Tester2 v_SLAC_Tester2;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
   preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

   //-------------- Test behavior------------------------------------------------------
   if( preConVerdict == pass ) {
        v_SLAC_Tester2.
        start(f_EVCC_CMN_TB_VTB_CmValidate_012(v_HAL_61851_Listener,
                                               par_cmValidate_result_failure));
        var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        if(testbehaviorVerdict == pass) {
            testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
        }
        if(testbehaviorVerdict == pass) {
            f_EVCC_CMN_TB_VTB_CmValidate_006(v_HAL_61851_Listener,
                                             par_cmValidate_result_failure);
        }
        v_SLAC_Tester2.done;
        if(getverdict == pass) {
            setverdict(pass, "SUT has continued the SLAC validation process " &
                             "with the next potential EVSE");
        }
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-----------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_011() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var SLAC_Tester2 v_SLAC_Tester2;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions----------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior------------------------------------------------------
    if( preConVerdict == pass ) {
        v_SLAC_Tester2.
        start(f_EVCC_CMN_TB_VTB_CmValidate_012(v_HAL_61851_Listener,
                                               par_cmValidate_result_notReady));
        var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        if(testbehaviorVerdict == pass) {
            testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
        }
        if(testbehaviorVerdict == pass) {
            f_EVCC_CMN_TB_VTB_CmValidate_006(v_HAL_61851_Listener,
                                             par_cmValidate_result_notReady);
        }
        v_SLAC_Tester2.done;
        if(getverdict == pass) {
            setverdict(pass, "SUT has continued the SLAC validation process " &
                             "with the next potential EVSE");
        }
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-----------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_012() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var SLAC_Tester2 v_SLAC_Tester2;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions----------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior------------------------------------------------------
    if( preConVerdict == pass ) {
        v_SLAC_Tester2.
        start(f_EVCC_CMN_TB_VTB_CmValidate_012(v_HAL_61851_Listener,
                                               par_cmValidate_result_notRequired));
        var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        if(testbehaviorVerdict == pass) {
            testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
        }
        if(testbehaviorVerdict == pass) {
            f_EVCC_CMN_TB_VTB_CmValidate_006(v_HAL_61851_Listener,
                                             par_cmValidate_result_notRequired);
        }
        v_SLAC_Tester2.done;
        if(getverdict == pass) {
            setverdict(pass, "SUT has continued the SLAC validation process " &
                             "with the next potential EVSE");
        }
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------ Post Conditions-----------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_013() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var SLAC_Tester2 v_SLAC_Tester2;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
    preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

    //-------------- Test behavior------------------------------------------------------
    if( preConVerdict == pass ) {
        v_SLAC_Tester2.start(f_EVCC_CMN_TB_VTB_CmValidate_010(par_cmValidate_result_failure));
        var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        if(testbehaviorVerdict == pass) {
            testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
        }
        if(testbehaviorVerdict == pass) {
            f_EVCC_CMN_TB_VTB_CmValidate_004(par_cmValidate_result_failure, true);
        }
        v_SLAC_Tester2.done;
        if(getverdict == pass) {
            setverdict(pass, "SUT has continued the SLAC validation process " &
                             "with the next potential EVSE");
        }
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------ Post Conditions-----------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_014() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-----------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

    //-------------- Test behavior------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_CmValidate_004(par_cmValidate_result_failure, true);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------ Post Conditions-----------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmValidate_015() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
```

```
      var verdicttype preConVerdict;

      // -------------- Pre Conditions-------------------------------------------------------
      f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

      //-------------- Test behavior--------------------------------------------------------
      if( preConVerdict == pass ) {
          f_EVCC_CMN_TB_VTB_CmValidate_004(par_cmValidate_result_failure, false);
          if(getverdict == pass) {
              f_EVCC_CMN_TB_VTB_CmSlacMatch_001();
          }
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions-------------------------------------------------------
      f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

   testcase TC_EVCC_CMN_VTB_CmValidate_016() runs on EVCC_Tester system SystemEVCC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions-------------------------------------------------------
      f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

      //-------------- Test behavior--------------------------------------------------------
      if( preConVerdict == pass ) {
          f_EVCC_CMN_TB_VTB_CmValidate_007();
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions-------------------------------------------------------
      f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

   testcase TC_EVCC_CMN_VTB_CmValidate_017() runs on EVCC_Tester system SystemEVCC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions-------------------------------------------------------
      f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

      //-------------- Test behavior--------------------------------------------------------
      if( preConVerdict == pass ) {
          f_EVCC_CMN_TB_VTB_CmValidate_004(par_cmValidate_result_notRequired, false);
          if(getverdict == pass) {
              f_EVCC_CMN_TB_VTB_CmSlacMatch_001();
          }
      } else {
          log("PreCondition was unsuccessful.");
      }

      //------------- Post Conditions-------------------------------------------------------
      f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
      f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   }

   testcase TC_EVCC_CMN_VTB_CmValidate_018() runs on EVCC_Tester system SystemEVCC {

      var HAL_61851_Listener v_HAL_61851_Listener;
      var verdicttype preConVerdict;

      // -------------- Pre Conditions-------------------------------------------------------
      f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

      //-------------- Test behavior--------------------------------------------------------
      if( preConVerdict == pass ) {
          f_EVCC_CMN_TB_VTB_CmValidate_008();
      } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmValidate_019() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var SLAC_Tester2 v_SLAC_Tester2;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions----------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
        preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            v_SLAC_Tester2.start(f_EVCC_CMN_TB_VTB_CmValidate_010(par_cmValidate_result_notReady));
            var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
            if(testbehaviorVerdict == pass) {
                testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
            }
            if(testbehaviorVerdict == pass) {
                f_EVCC_CMN_TB_VTB_CmValidate_004(par_cmValidate_result_notReady, true);
            }
            v_SLAC_Tester2.done;
            if(getverdict == pass) {
                setverdict(pass, "SUT has continued the SLAC validation process " &
                                 "with the next potential EVSE");
            }
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
    }

    testcase TC_EVCC_CMN_VTB_CmValidate_020() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions----------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmValidate_013(e_OscOff);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmValidate_021() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions----------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmValidate_013(e_NegVolt12);
        } else {
            log("PreCondition was unsuccessful.");
        }
        //------------- Post Conditions---------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }
}
```

C.2.4 EVCC test cases for CmValidateOrCmSlacMatch

```
module TestCases_EVCC_CmValidateOrCmSlacMatch {

    import from DataStructure_SLAC all;
    import from TestBehavior_EVCC_AttenuationCharacterization all;
    import from TestBehavior_EVCC_CmValidateOrCmSlacMatch all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from Timer_15118_3 all;
    import from PreConditions_EVCC_15118_3 all;
    import from PostConditions_EVCC_15118_3 all;
    import from Templates_CMN_CmSlacMatch all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from TestBehavior_EVCC_AttenuationCharacterization all;
    import from TestBehavior_EVCC_CmSlacParm all;

    testcase TC_EVCC_CMN_VTB_CmValidateOrCmSlacMatch_001() runs on EVCC_Tester system SystemEVCC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions---------------------------------------------------------
       f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

       //-------------- Test behavior----------------------------------------------------------
       if( preConVerdict == pass ) {
           f_EVCC_CMN_TB_VTB_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener, fail);
       } else {
           log("PreCondition was unsuccessful.");
       }


       //------------- Post Conditions---------------------------------------------------------
       f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }
}
```

C.2.5 EVCC test cases for CmSlacMatch

```
module TestCases_EVCC_CmSlacMatch {

    import from DataStructure_SLAC all;
    import from TestBehavior_EVCC_CmSlacMatch all;
    import from TestBehavior_EVCC_CmValidateOrCmSlacMatch all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from Timer_15118_3 all;
    import from PreConditions_EVCC_15118_3 all;
    import from PostConditions_EVCC_15118_3 all;
    import from Templates_CMN_CmSlacMatch all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from TestBehavior_EVCC_AttenuationCharacterization all;
    import from TestBehavior_EVCC_CmSlacParm all;
    import from LibFunctions_15118_3  { group generalFunctions; }

    testcase TC_EVCC_CMN_VTB_CmSlacMatch_001() runs on EVCC_Tester system SystemEVCC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions---------------------------------------------------------
       f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_EVCC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);

       //-------------- Test behavior----------------------------------------------------------
       if( preConVerdict == pass ) {
           f_EVCC_CMN_TB_VTB_CmSlacMatch_001();
```

```
   } else {
      log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions----------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

 testcase TC_EVCC_CMN_VTB_CmSlacMatch_002() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_CmSlacMatch_001();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

testcase TC_EVCC_CMN_VTB_CmSlacMatch_003() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_002();
   } else {
      log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacMatch_004() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior---------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                  m_CMN_CMN_SlacPayloadHeaderInvalid_001(),
                                  '0056'H,
                                  '00000000000000000000000000000000'H,
                                  vc_sut_mac,
                                  '00000000000000000000000000000000'H,
                                  par_testSystem_mac,
                                  vc_RunID, vc_Nid, vc_Nmk));
   } else {
      log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions--------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}
```

```
testcase TC_EVCC_CMN_VTB_CmSlacMatch_005() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                          m_CMN_CMN_SlacPayloadHeaderInvalid_002(),
                                          '0056'H,
                                          '00000000000000000000000000000000'H,
                                          vc_sut_mac,
                                          '00000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          vc_RunID, vc_Nid, vc_Nmk));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-----------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacMatch_006() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                          m_CMN_CMN_SlacPayloadHeader_001(),
                                          '0000'H,
                                          '00000000000000000000000000000000'H,
                                          vc_sut_mac,
                                          '00000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          vc_RunID, vc_Nid, vc_Nmk));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-----------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacMatch_007() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                          m_CMN_CMN_SlacPayloadHeader_001(),
                                          '0056'H,
                                          '00000000000000000000000000000001'H,
                                          vc_sut_mac,
                                          '00000000000000000000000000000000'H,
                                          par_testSystem_mac,
                                          vc_RunID, vc_Nid, vc_Nmk));
   } else {
       log("PreCondition was unsuccessful.");
   }
   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacMatch_008() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                         m_CMN_CMN_SlacPayloadHeader_001(),
                                         '0056'H,
                                         '0000000000000000000000000000000000'H,
                                         '000000000000'H,
                                         '0000000000000000000000000000000000'H,
                                         par_testSystem_mac,
                                         vc_RunID, vc_Nid, vc_Nmk));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacMatch_009() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                         m_CMN_CMN_SlacPayloadHeader_001(),
                                         '0056'H,
                                         '0000000000000000000000000000000000'H,
                                         vc_sut_mac,
                                         '0000000000000000000000000000000001'H,
                                         par_testSystem_mac,
                                         vc_RunID, vc_Nid, vc_Nmk));
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions---------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_CmSlacMatch_010() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions--------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

   //-------------- Test behavior----------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                         m_CMN_CMN_SlacPayloadHeader_001(),
                                        '0056'H,
                                        '0000000000000000000000000000000000'H,
                                        vc_sut_mac,
                                          '00000000000000000000000000000000'H,
                                          '000000000000'H,
                                          vc_RunID, vc_Nid, vc_Nmk));
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmSlacMatch_011() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions--------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            var RunID_TYPE v_RunID := f_randomHexStringGen(16);
            if(v_RunID != vc_RunID) {

                f_EVCC_CMN_TB_VTB_CmSlacMatch_003(md_CMN_CMN_CmSlacMatchCnf_002(
                                                  m_CMN_CMN_SlacPayloadHeader_001(),
                                                  '0056'H,
                                                  '00000000000000000000000000000000'H,
                                                  vc_sut_mac,
                                                  '00000000000000000000000000000000'H,
                                                  par_testSystem_mac,
                                                  v_RunID, vc_Nid, vc_Nmk));
            } else {setverdict(inconc, "Invalid runID is equal to current runID.")};
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmSlacMatch_012() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var SLAC_Tester2 v_SLAC_Tester2;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions--------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
        preConVerdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------------
        if( preConVerdict == pass ) {
            v_SLAC_Tester2.start(f_EVCC_CMN_TB_VTB_AttenuationCharacterization_003());
            var verdicttype testbehaviorVerdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
            if(testbehaviorVerdict == pass) {
                testbehaviorVerdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(fail);
            }
            if(testbehaviorVerdict == pass) {
                f_EVCC_CMN_TB_VTB_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener, fail);
            }
            v_SLAC_Tester2.done;
        } else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(v_HAL_61851_Listener, v_SLAC_Tester2, system);
    }
}
```

C.2.6 EVCC test cases for PLCLinkStatus

```
module TestCases_EVCC_PLCLinkStatus {

    import from DataStructure_SLAC all;
    import from TestBehavior_EVCC_PLCLinkStatus all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_EVCC_15118_3 all;
    import from Timer_15118_3 all;
    import from Pixit_15118_3 all;
    import from PostConditions_EVCC_15118_3 all;
    import from PreConditions_EVCC_15118_2 {
        function f_EVCC_AC_PR_SessionStop_002,
        f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002,
        f_EVCC_AC_PR_SessionStop_003,
        f_EVCC_DC_PR_WeldingDetectionOrSessionStop_003
    };
    import from Configurations_15118_2 all;
    import from LibFunctions_15118_3 all;
    import from Services_HAL_61851 all;
    import from TestBehavior_EVCC_CommonBehavior all;
    import from Timer_15118 all;
    import from Pixit_15118 all;
    import from Pics_15118 all;
    import from Timer_15118_2 all;

    testcase TC_EVCC_CMN_VTB_PLCLinkStatus_001() runs on EVCC_Tester system SystemEVCC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-----------------------------------------------------
       f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

       //-------------- Test behavior------------------------------------------------------
       if( preConVerdict == pass ) {
           f_EVCC_CMN_TB_VTB_PLCLinkStatus_001(fail);
       } else {
           log("PreCondition was unsuccessful.");
       }

       //------------- Post Conditions-----------------------------------------------------
       f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
       f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_EVCC_CMN_VTB_PLCLinkStatus_002() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-----------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_PLCLinkStatus_002();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-----------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       }

     testcase TC_EVCC_CMN_VTB_PLCLinkStatus_003() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-----------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

        //-------------- Test behavior------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_PLCLinkStatus_003(e_OscOff);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_PLCLinkStatus_004() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions---------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

        //-------------- Test behavior-----------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_PLCLinkStatus_003(e_NegVolt12);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_PLCLinkStatus_005() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions---------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        vc_DutyCycleDelay := 7.5;
        preConVerdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);

        //-------------- Test behavior-----------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_PLCLinkStatus_001(fail);
        } else {
            log("PreCondition was unsuccessful.");
        }
        if(getverdict == pass) {
            log("A SECC delay of 7,5 s for signaling a 5% duty cycle did not " &
                "influence the SLAC Matching process.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_PLCLinkStatus_006() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions---------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior-----------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_PLCLinkStatus_006(v_HAL_61851_Listener,5, E);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_PLCLinkStatus_007() runs on EVCC_Tester system SystemEVCC {
```

```
   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

   //-------------- Test behavior--------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_PLCLinkStatus_006(v_HAL_61851_Listener,5, F);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_CMN_VTB_PLCLinkStatus_008() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions-------------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
   preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

   //-------------- Test behavior--------------------------------------------------------
   if( preConVerdict == pass ) {
       f_EVCC_CMN_TB_VTB_PLCLinkStatus_008(v_HAL_61851_Listener);
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions-------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_AC_VTB_PLCLinkStatus_001() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
    preConVerdict := f_EVCC_AC_PR_SessionStop_002(v_HAL_61851_Listener);

    if(preConVerdict == pass) {
       preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
    }

    if(preConVerdict == pass) {
       map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
       f_EVCC_startSleepingPhase(PICS_CMN_CMN_WakeUp);
    }

    //-------------- Test behavior--------------------------------------------------------
    if(preConVerdict == pass) {
       f_EVCC_CMN_TB_VTB_PLCLinkStatus_004(v_HAL_61851_Listener);
    }
    else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions-------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
}

testcase TC_EVCC_AC_VTB_PLCLinkStatus_002() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
```

```
     f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
     preConVerdict := f_EVCC_AC_PR_SessionStop_002(v_HAL_61851_Listener);

     if(preConVerdict == pass) {
        preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
     }

     if(preConVerdict == pass) {
        map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
        f_EVCC_startSleepingPhase(PIXIT_CMN_CMN_WakeUp - 5.0);
     }

     //-------------- Test behavior---------------------------------------------------------
     if(preConVerdict == pass) {
        f_EVCC_CMN_TB_VTB_PLCLinkStatus_005(v_HAL_61851_Listener, PIXIT_CMN_CMN_WakeUp);
     }
     else {
         log("PreCondition was unsuccessful.");
     }

     //------------- Post Conditions--------------------------------------------------------
     f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
     f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
 }

 testcase TC_EVCC_AC_VTB_PLCLinkStatus_003() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_PLCLinkStatus_006(v_HAL_61851_Listener,
                                            par_EVSENominalDutyCycle, E);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

 testcase TC_EVCC_AC_VTB_PLCLinkStatus_004() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_AC_TB_VTB_PLCLinkStatus_001();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
 }

 testcase TC_EVCC_AC_VTB_PLCLinkStatus_005() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior---------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_AC_TB_VTB_PLCLinkStatus_002(v_HAL_61851_Listener, E);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_AC_VTB_PLCLinkStatus_006() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
    preConVerdict := f_EVCC_AC_PR_SessionStop_003(v_HAL_61851_Listener);

    //-------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
        f_EVCC_CMN_TB_VTB_PLCLinkStatus_007();
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_AC_VTB_PLCLinkStatus_007() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_CMN_TB_VTB_PLCLinkStatus_006(v_HAL_61851_Listener,
                                            par_EVSENominalDutyCycle, F);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_AC_VTB_PLCLinkStatus_008() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions-------------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

    //-------------- Test behavior--------------------------------------------------------
    if( preConVerdict == pass ) {
        f_EVCC_AC_TB_VTB_PLCLinkStatus_002(v_HAL_61851_Listener, F);
    } else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions--------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_AC_VTB_PLCLinkStatus_009() runs on EVCC_Tester system SystemEVCC {

     var HAL_61851_Listener v_HAL_61851_Listener;
```

```
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
        preConVerdict := f_EVCC_AC_PR_SessionStop_002(v_HAL_61851_Listener);

        if(preConVerdict == pass) {
           preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
        }

        if(preConVerdict == pass) {
           map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
           f_EVCC_startSleepingPhase(par_SECC_Pmax0W - 5.0);
        }

        //-------------- Test behavior---------------------------------------------------------
        if(preConVerdict == pass) {
           f_EVCC_CMN_TB_VTB_PLCLinkStatus_005(v_HAL_61851_Listener, par_SECC_Pmax0W);
        }
        else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
    }

    testcase TC_EVCC_AC_VTB_PLCLinkStatus_010() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
        preConVerdict := f_EVCC_AC_PR_SessionStop_002(v_HAL_61851_Listener);

        if(preConVerdict == pass) {
           preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
        }

        if(preConVerdict == pass) {
           map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
           f_EVCC_startSleepingPhase(par_SECC_Pmax0W - 5.0);
        }

        //-------------- Test behavior---------------------------------------------------------
        if(preConVerdict == pass) {
           f_EVCC_CMN_TB_VTB_PLCLinkStatus_005(v_HAL_61851_Listener, par_SECC_Pmax0W);
        }
        else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
    }

    testcase TC_EVCC_DC_VTB_PLCLinkStatus_001() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
        preConVerdict := f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002(
                                              v_HAL_61851_Listener);

        if(preConVerdict == pass) {
           preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
        }

        if(preConVerdict == pass) {
           map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
           f_EVCC_startSleepingPhase(PICS_CMN_CMN_WakeUp);
        }

        //-------------- Test behavior---------------------------------------------------------
    if(preConVerdict == pass) {
       f_EVCC_CMN_TB_VTB_PLCLinkStatus_004(v_HAL_61851_Listener);
    }
    else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
}

testcase TC_EVCC_DC_VTB_PLCLinkStatus_002() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions----------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
    preConVerdict := f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002(
                      v_HAL_61851_Listener);

    if(preConVerdict == pass) {
       preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
    }

    if(preConVerdict == pass) {
       map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
       f_EVCC_startSleepingPhase(PIXIT_CMN_CMN_WakeUp - 5.0);
    }

    //-------------- Test behavior------------------------------------------------------
    if(preConVerdict == pass) {
       f_EVCC_CMN_TB_VTB_PLCLinkStatus_005(v_HAL_61851_Listener, PIXIT_CMN_CMN_WakeUp);
    }
    else {
        log("PreCondition was unsuccessful.");
    }

    //------------- Post Conditions------------------------------------------------------
    f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
    f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
}

testcase TC_EVCC_DC_VTB_PLCLinkStatus_003() runs on EVCC_Tester system SystemEVCC {

   var HAL_61851_Listener v_HAL_61851_Listener;
   var verdicttype preConVerdict;

   // -------------- Pre Conditions----------------------------------------------------
   f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
   preConVerdict := f_EVCC_DC_PR_WeldingDetectionOrSessionStop_003(v_HAL_61851_Listener);

   //-------------- Test behavior------------------------------------------------------
   if( preConVerdict == pass ) {
       map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
       f_EVCC_CMN_TB_VTB_PLCLinkStatus_007();
   } else {
       log("PreCondition was unsuccessful.");
   }

   //------------- Post Conditions------------------------------------------------------
   f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
   f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener, system);
}

testcase TC_EVCC_DC_VTB_PLCLinkStatus_004() runs on EVCC_Tester system SystemEVCC {

    var HAL_61851_Listener v_HAL_61851_Listener;
    var verdicttype preConVerdict;

    // -------------- Pre Conditions----------------------------------------------------
    f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
    preConVerdict := f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002(
                     v_HAL_61851_Listener);

    if(preConVerdict == pass) {
       preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
```

```
        }

        if(preConVerdict == pass) {
           map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
           f_EVCC_startSleepingPhase(par_SECC_Pmax0W - 5.0);
        }

        //-------------- Test behavior---------------------------------------------------------
        if(preConVerdict == pass) {
           f_EVCC_CMN_TB_VTB_PLCLinkStatus_005(v_HAL_61851_Listener, par_SECC_Pmax0W);
        }
        else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
    }

    testcase TC_EVCC_DC_VTB_PLCLinkStatus_005() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-----------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_001(v_HAL_61851_Listener,system);
        preConVerdict := f_EVCC_DC_PR_WeldingDetectionOrSessionStop_002(
                         v_HAL_61851_Listener);

        if(preConVerdict == pass) {
           preConVerdict := f_EVCC_setPwmMode(e_PosVolt12);
        }

        if(preConVerdict == pass) {
           map(mtc:pt_SLAC_Port, system:pt_SLAC_Port);
           f_EVCC_startSleepingPhase(par_SECC_Pmax0W - 5.0);
        }

        //-------------- Test behavior---------------------------------------------------------
        if(preConVerdict == pass) {
           f_EVCC_CMN_TB_VTB_PLCLinkStatus_005(v_HAL_61851_Listener, par_SECC_Pmax0W);
        }
        else {
            log("PreCondition was unsuccessful.");
        }


        //------------- Post Conditions------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_001(v_HAL_61851_Listener,system);
    }
}
```

C.2.7 EVCC test cases for CmAmpMap

```
module TestCases_EVCC_CmAmpMap {

    import from DataStructure_SLAC all;
    import from TestBehavior_EVCC_CmAmpMap all;
    import from TestBehavior_EVCC_PLCLinkStatus all;
    import from ComponentsAndPorts all;
    import from Configurations_15118_3 all;
    import from PreConditions_EVCC_15118_3 all;
    import from Timer_15118_3 all;
    import from PostConditions_EVCC_15118_3 all;
    import from Services_TXPowerLimitation all;

   testcase TC_EVCC_CMN_VTB_CmAmpMap_001() runs on EVCC_Tester system SystemEVCC {

       var HAL_61851_Listener v_HAL_61851_Listener;
       var verdicttype preConVerdict;

       // -------------- Pre Conditions-----------------------------------------------------
       f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
       preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

       //-------------- Test behavior---------------------------------------------------------
```

```
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmAmpMap_001(fail);
            if(getverdict == pass) {
              f_EVCC_CMN_checkTXPowerLimitation();
            }
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmAmpMap_002() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior---------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmAmpMap_002(fail);
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions--------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

     testcase TC_EVCC_CMN_VTB_CmAmpMap_003() runs on EVCC_Tester system SystemEVCC {

         var HAL_61851_Listener v_HAL_61851_Listener;
         var verdicttype preConVerdict;

         // -------------- Pre Conditions-------------------------------------------------
         f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
         preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

         //-------------- Test behavior---------------------------------------------------
         if( preConVerdict == pass ) {
             f_EVCC_CMN_TB_VTB_CmAmpMap_003();
         } else {
             log("PreCondition was unsuccessful.");
         }

         //------------- Post Conditions--------------------------------------------------
         f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
         f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
     }

     testcase TC_EVCC_CMN_VTB_CmAmpMap_004() runs on EVCC_Tester system SystemEVCC {

         var HAL_61851_Listener v_HAL_61851_Listener;
         var verdicttype preConVerdict;

         // -------------- Pre Conditions-------------------------------------------------
         f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
         preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

         //-------------- Test behavior---------------------------------------------------
         if( preConVerdict == pass ) {
             f_EVCC_CMN_TB_VTB_CmAmpMap_004();
         } else {
             log("PreCondition was unsuccessful.");
         }

         //------------- Post Conditions--------------------------------------------------
         f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
         f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
      }

     testcase TC_EVCC_CMN_VTB_CmAmpMap_005() runs on EVCC_Tester system SystemEVCC {
        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior-------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmAmpMap_005();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmAmpMap_006() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior-------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmAmpMap_006();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmAmpMap_007() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

        //-------------- Test behavior-------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmAmpMap_007();
        } else {
            log("PreCondition was unsuccessful.");
        }

        //------------- Post Conditions-------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }

    testcase TC_EVCC_CMN_VTB_CmAmpMap_008() runs on EVCC_Tester system SystemEVCC {

        var HAL_61851_Listener v_HAL_61851_Listener;
        var verdicttype preConVerdict;

        // -------------- Pre Conditions-------------------------------------------------------
        f_EVCC_CMN_PR_InitConfiguration_SLAC_001(v_HAL_61851_Listener, system);
        preConVerdict := f_EVCC_CMN_PR_CmAmpMap_001(v_HAL_61851_Listener);

        //-------------- Test behavior-------------------------------------------------------
        if( preConVerdict == pass ) {
            f_EVCC_CMN_TB_VTB_CmAmpMap_008();
        } else {
            log("PreCondition was unsuccessful.");
        }
        //------------- Post Conditions--------------------------------------------------------
        f_EVCC_CMN_PO_InitialState_001(v_HAL_61851_Listener);
        f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(v_HAL_61851_Listener, system);
    }
}
```

Annex D
(normative)

Function specifications for supporting test execution

D.1 Configuration functions

```
module Configurations_15118_3 {

    import from Services_HAL_61851 all;
    import from ComponentsAndPorts all;
    import from LibFunctions_15118_3 { group generalFunctions; }
    import from Pics_15118 all;

    //     :::::::::::::::::::::: Config Functions :::::::::::::::::::::::

    function f_SECC_CMN_PR_InitConfiguration_SLAC_001(out HAL_61851_Listener v_HAL_61851_Listener,
                                                      SystemSECC v_SystemSECC) runs on SECC_Tester {

        map(mtc:pt_SLAC_Port, v_SystemSECC:pt_SLAC_Port);

            v_HAL_61851_Listener := HAL_61851_Listener.create("IEC 61851 Listener") alive;

            map(mtc:pt_HAL_61851_Port, v_SystemSECC:pt_HAL_61851_Port);
            map(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
                v_SystemSECC:pt_HAL_61851_Listener_Port);
            connect(mtc:pt_HAL_61851_Internal_Port, v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);

            v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(false));

            activate(a_CMN_IEC61851ListenerBehavior(pt_HAL_61851_Internal_Port));
     }

    function f_SECC_CMN_PR_InitConfiguration_SLAC_002(out HAL_61851_Listener v_HAL_61851_Listener,
                                                      out SLAC_Tester2 v_SLAC_Tester2,
                                                      out SLAC_Tester3 v_SLAC_Tester3,
                                                      out SLAC_Tester4 v_SLAC_Tester4,
                                                      out SLAC_Tester5 v_SLAC_Tester5,
                                                      SystemSECC systemSECC) runs on SECC_Tester  {

        var hexstring emptyMacAddress := '000000000000'H;
        map(mtc:pt_SLAC_Port, systemSECC:pt_SLAC_Port);

        if(par_slac_node2_mac != emptyMacAddress) {
            v_SLAC_Tester2 := SLAC_Tester2.create("Slac Tester 2") alive;
            map(v_SLAC_Tester2:pt_SLAC_Port, systemSECC:pt_SLAC_Port) param (par_slac_node2_mac);
        }
        else {log("MAC address of Slac node 2 is empty.");}
        if(par_slac_node3_mac != emptyMacAddress) {
            v_SLAC_Tester3 := SLAC_Tester3.create("Slac Tester 3") alive;
            map(v_SLAC_Tester3:pt_SLAC_Port, systemSECC:pt_SLAC_Port) param (par_slac_node3_mac);
        }
        else {log("MAC address of Slac node 3 is empty.");}
        if(par_slac_node4_mac != emptyMacAddress) {
            v_SLAC_Tester4 := SLAC_Tester4.create("Slac Tester 4") alive;
            map(v_SLAC_Tester4:pt_SLAC_Port, systemSECC:pt_SLAC_Port) param (par_slac_node4_mac);
        }
        else {log("MAC address of Slac node 4 is empty.");}
        if(par_slac_node5_mac != emptyMacAddress) {
            v_SLAC_Tester5 := SLAC_Tester5.create("Slac Tester 5") alive;
            map(v_SLAC_Tester5:pt_SLAC_Port, systemSECC:pt_SLAC_Port) param (par_slac_node5_mac);
        }
        else {log("MAC address of Slac node 5 is empty.");}

            v_HAL_61851_Listener := HAL_61851_Listener.create("IEC 61851 Listener") alive;
            map(mtc:pt_HAL_61851_Port, systemSECC:pt_HAL_61851_Port);
            map(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
                systemSECC:pt_HAL_61851_Listener_Port);
            connect(mtc:pt_HAL_61851_Internal_Port,
                    v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);

            v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(false));
        vc_Default_IEC_61851_ListenerBehavior := activate(a_CMN_IEC61851ListenerBehavior(
                                                  pt_HAL_61851_Internal_Port));
 }

 function f_SECC_CMN_PO_ShutdownConfiguration_SLAC_001(out HAL_61851_Listener
                                                       v_HAL_61851_Listener,
                                                       SystemSECC v_SystemSECC)
                                                       runs on SECC_Tester  {

        unmap(mtc:pt_HAL_61851_Port, v_SystemSECC:pt_HAL_61851_Port);
        unmap(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
              v_SystemSECC:pt_HAL_61851_Listener_Port);
        disconnect(mtc:pt_HAL_61851_Internal_Port,
                   v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);

        v_HAL_61851_Listener.kill;


    unmap(mtc:pt_SLAC_Port, v_SystemSECC:pt_SLAC_Port);
  }

 function f_SECC_CMN_PO_ShutdownConfiguration_SLAC_002(out HAL_61851_Listener
                                                       v_HAL_61851_Listener,
                                                       out SLAC_Tester2 v_SLAC_Tester2,
                                                       out SLAC_Tester3 v_SLAC_Tester3,
                                                       out SLAC_Tester4 v_SLAC_Tester4,
                                                       out SLAC_Tester5 v_SLAC_Tester5,
                                                       SystemSECC v_SystemSECC)
                                                       runs on SECC_Tester  {

        unmap(mtc:pt_HAL_61851_Port, v_SystemSECC:pt_HAL_61851_Port);
        unmap(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
              v_SystemSECC:pt_HAL_61851_Listener_Port);
        disconnect(mtc:pt_HAL_61851_Internal_Port,
                   v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);


    unmap(mtc:pt_SLAC_Port, v_SystemSECC:pt_SLAC_Port);
    if(v_SLAC_Tester2.running) {
        unmap(v_SLAC_Tester2:pt_SLAC_Port, v_SystemSECC:pt_SLAC_Port);
    }
    if(v_SLAC_Tester3.running) {
        unmap(v_SLAC_Tester3:pt_SLAC_Port, v_SystemSECC:pt_SLAC_Port);
    }
    if(v_SLAC_Tester4.running) {
        unmap(v_SLAC_Tester4:pt_SLAC_Port, v_SystemSECC:pt_SLAC_Port);
    }
    if(v_SLAC_Tester5.running) {
        unmap(v_SLAC_Tester5:pt_SLAC_Port, v_SystemSECC:pt_SLAC_Port);
    }

    all component.kill;
 }

 //      ::::::::::::::::::::::: Config Functions ::::::::::::::::::::::::

function f_EVCC_CMN_PR_InitConfiguration_SLAC_001(out HAL_61851_Listener v_HAL_61851_Listener,
                                                  SystemEVCC v_SystemEVCC) runs on EVCC_Tester {

    map(mtc:pt_SLAC_Port, v_SystemEVCC:pt_SLAC_Port);

        v_HAL_61851_Listener := HAL_61851_Listener.create("IEC 61851 Listener") alive;

        map(mtc:pt_HAL_61851_Port, v_SystemEVCC:pt_HAL_61851_Port);
        map(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
            v_SystemEVCC:pt_HAL_61851_Listener_Port);
        connect(mtc:pt_HAL_61851_Internal_Port,
                v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);

        v_HAL_61851_Listener.start(f_EVCC_HAL61851Listener(false));

        activate(a_CMN_IEC61851ListenerBehavior(pt_HAL_61851_Internal_Port));
}

function f_EVCC_CMN_PR_InitConfiguration_SLAC_002(out HAL_61851_Listener v_HAL_61851_Listener,
                                                  out SLAC_Tester2 v_SLAC_Tester2,
                                                  SystemEVCC systemEVCC) runs on EVCC_Tester {
```

```
        var hexstring emptyMacAddress := '000000000000'H;
        map(mtc:pt_SLAC_Port, systemEVCC:pt_SLAC_Port);

        if(par_slac_node2_mac != emptyMacAddress or isbound(v_SLAC_Tester2)) {
            v_SLAC_Tester2 := SLAC_Tester2.create("Slac Tester 2") alive;
            map(v_SLAC_Tester2:pt_SLAC_Port, systemEVCC:pt_SLAC_Port) param (par_slac_node2_mac);
        }
        else {log("MAC address of Slac node 2 is empty.");}

            v_HAL_61851_Listener := HAL_61851_Listener.create("IEC 61851 Listener") alive;

            map(mtc:pt_HAL_61851_Port, systemEVCC:pt_HAL_61851_Port);
            map(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
                systemEVCC:pt_HAL_61851_Listener_Port);
            connect(mtc:pt_HAL_61851_Internal_Port, v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);

            v_HAL_61851_Listener.start(f_EVCC_HAL61851Listener(false));

            vc_Default_IEC_61851_ListenerBehavior := activate(a_CMN_IEC61851ListenerBehavior(
                                                             pt_HAL_61851_Internal_Port));
    }

    function f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_001(out HAL_61851_Listener
                                                             v_HAL_61851_Listener,
                                                          SystemEVCC v_SystemEVCC)
                                                          runs on EVCC_Tester  {


             unmap(mtc:pt_HAL_61851_Port, v_SystemEVCC:pt_HAL_61851_Port);
             unmap(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
                   v_SystemEVCC:pt_HAL_61851_Listener_Port);
             disconnect(mtc:pt_HAL_61851_Internal_Port,
                        v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);

             v_HAL_61851_Listener.kill;


       unmap(mtc:pt_SLAC_Port, v_SystemEVCC:pt_SLAC_Port);
    }

    function f_EVCC_CMN_PO_ShutdownConfiguration_SLAC_002(out HAL_61851_Listener
                                                             v_HAL_61851_Listener,
                                                          out SLAC_Tester2 v_SLAC_Tester2,
                                                          SystemEVCC v_SystemEVCC)
                                                          runs on EVCC_Tester  {

            unmap(mtc:pt_HAL_61851_Port, v_SystemEVCC:pt_HAL_61851_Port);
            unmap(v_HAL_61851_Listener:pt_HAL_61851_Listener_Port,
                  v_SystemEVCC:pt_HAL_61851_Listener_Port);
            disconnect(mtc:pt_HAL_61851_Internal_Port,
                       v_HAL_61851_Listener:pt_HAL_61851_Internal_Port);



        unmap(mtc:pt_SLAC_Port, v_SystemEVCC:pt_SLAC_Port);
        unmap(v_SLAC_Tester2:pt_SLAC_Port, v_SystemEVCC:pt_SLAC_Port);

        all component.kill;
    }
}
```

D.2 Pre-condition functions

D.2.1 SECC + PLC bridge functions

```
module PreConditions_SECC_15118_3 {

    import from TestBehavior_SECC_CmSlacParm all;
    import from TestBehavior_SECC_AttenuationCharacterization all;
    import from TestBehavior_SECC_CmSlacMatch all;
    import from TestBehavior_SECC_CmSetKey all;
    import from TestBehavior_SECC_PLCLinkStatus all;
    import from TestBehavior_SECC_CmValidate all;
    import from TestBehavior_SECC_CmAmpMap all;
    import from Timer_15118_3 all;
    import from ComponentsAndPorts all;
import from LibFunctions_15118_3 all;
import from Services_HAL_61851 all;
import from Pics_15118_3 all;
import from Pixit_15118_3 all;
import from Pics_15118 all;
import from Services_PLCLinkStatus all;
import from Timer_15118 all;
import from Services_EIMIdentification all;
import from Services_TXPowerLimitation all;

function f_SECC_CMN_PR_StateA_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                  runs on SECC_Tester return verdicttype {

    var verdicttype verdict := pass;

        f_SECC_changeValidStateCondition(A);
        verdict := f_SECC_setState(A,v_HAL_61851_Listener);
        pt_HAL_61851_Port.clear;
        sleep((par_CMN_waitForNextHAL));
        verdict := f_SECC_setProximity(0);

    if(PICS_SECC_CMN_EIMDone == beforePlugin and vc_testCaseSpecific) {
        var boolean v_result := f_SECC_CMN_EIMIdentification();
        if(v_result) {
            f_SECC_setEimStatus(v_result);
        }
    }
    return verdict;
}

//  SECC
function f_SECC_CMN_PR_DisconnectDataLink_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                              runs on SECC_Tester return verdicttype {

   var verdicttype verdict := f_SECC_CMN_PR_StateA_001(v_HAL_61851_Listener);

          // generate Nid and Nmk
       vc_Nmk := f_randomHexStringGen(32);
       vc_Nid := fx_generateNID(vc_Nmk);
       verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(true);
   return verdict;
}

function f_SECC_CMN_PR_SetProximityPilot_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                             runs on SECC_Tester return verdicttype {

    var verdicttype verdict := f_SECC_CMN_PR_DisconnectDataLink_001(v_HAL_61851_Listener);

        if ( verdict == pass ) {

            if(vc_sleepAfterPlugOut) {
                sleep((par_SECC_waitForPlugin));
            }

            v_HAL_61851_Listener.stop;
            v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(true));

            if(vc_activateNominal){
                f_SECC_changeValidFrequencyRange(0,0);
                f_SECC_changeValidDutyCycleRange(100,100);
                vc_validDutyCycleLowerBound2 := 100;
                vc_validDutyCycleUpperBound2 := 100;
            }
            if(PICS_CMN_CMN_PlugType == type1) {
                verdict := f_SECC_setProximity(cc_proximity_type1);
            } else {
                if(PICS_CMN_CMN_ChargingMode == aC){
                    var integer v_proximity_type2_AC;
                    if(PICS_CMN_AC_CableCapability == capability13A) {
                        v_proximity_type2_AC := cc_proximity_type2_AC_13A;
                    } else if(PICS_CMN_AC_CableCapability == capability20A) {
                        v_proximity_type2_AC := cc_proximity_type2_AC_20A;
                    } else if(PICS_CMN_AC_CableCapability == capability32A) {
                        v_proximity_type2_AC := cc_proximity_type2_AC_32A;
                    } else {
                        v_proximity_type2_AC := cc_proximity_type2_AC_63A;
                    }
                    verdict := f_SECC_setProximity(v_proximity_type2_AC);
                } else {
                    verdict := f_SECC_setProximity(cc_proximity_type2_DC);
                }
            }
        }
    return verdict;
}

function f_SECC_CMN_PR_StateB_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                runs on SECC_Tester return verdicttype {

    var verdicttype verdict := f_SECC_CMN_PR_SetProximityPilot_001(v_HAL_61851_Listener);


        if ( verdict == pass ) {
            sleep((par_CMN_waitForNextHAL));
            f_SECC_changeValidStateCondition(valid);

            f_SECC_setState(vc_state,v_HAL_61851_Listener);

            if(PICS_SECC_CMN_EIMDone == afterPlugin and vc_testCaseSpecific) {
                f_SECC_setIsConfirmationFlagDC();
                verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                                    par_T_conn_max_comm,
                                                    inconc);

                f_SECC_changeValidStateCondition(EorF);
                f_SECC_changeValidFrequencyRange(0,0);
                f_SECC_changeValidDutyCycleRange(0,0);
                f_SECC_setIsConfirmationFlagVoltage();
                var boolean v_result := f_SECC_CMN_EIMIdentification();
                if(v_result) {
                    f_SECC_setEimStatus(v_result);
                }
            } else if(PICS_SECC_CMN_EIMDone == duringSlac and vc_testCaseSpecific) {

                f_SECC_setIsConfirmationFlagDC();
                verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                                    par_T_conn_max_comm,
                                                    inconc);
                if(verdict == pass) {
                    if(PIXIT_SECC_AC_InitialDutyCyle == dc5) {
                        f_SECC_changeValidDutyCycleRange(100,100);
                        vc_validDutyCycleLowerBound2 := 100;
                        vc_validDutyCycleUpperBound2 := 100;
                    }
                    var boolean v_result := f_SECC_CMN_EIMIdentification();
                }
            }
        }
    tc_TT_matching_repetition.start(par_TT_matching_repetition);

    return verdict;
}

function f_SECC_CMN_PR_CmSlacParm_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                    runs on SECC_Tester return verdicttype {

   var verdicttype verdict := f_SECC_CMN_PR_StateB_001(v_HAL_61851_Listener);
   if ( verdict == pass ) {
       verdict := f_SECC_CMN_TB_VTB_CmSlacParm_001(inconc);
   }
   return verdict;
}

function f_SECC_CMN_PR_AttenuationCharacterization_001(out HAL_61851_Listener
                                                           v_HAL_61851_Listener)
                                                           runs on SECC_Tester
                                                           return verdicttype {

   var verdicttype verdict := f_SECC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);
  // SECC_AttenuationCharacterization Behavior
  if ( verdict == pass ) {
      verdict := f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(inconc);
  }
  return verdict;
}

function f_SECC_CMN_PR_CmValidate_001(out HAL_61851_Listener v_HAL_61851_Listener)
```

```
                                    runs on SECC_Tester return verdicttype {

       var verdicttype verdict := f_SECC_CMN_PR_AttenuationCharacterization_001(
                                  v_HAL_61851_Listener);
      // SECC_CmSlacMatch Behavior
      if ( verdict == pass) {
          verdict := f_SECC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, inconc);
      }
      return verdict;
   }

   function f_SECC_CMN_PR_CmSlacMatch_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                    runs on SECC_Tester return verdicttype {

      var verdicttype verdict;

      if(PIXIT_SECC_CMN_CmValidate == cmValidate) {
          verdict := f_SECC_CMN_PR_CmValidate_001(v_HAL_61851_Listener);
      }
      else {
          verdict := f_SECC_CMN_PR_AttenuationCharacterization_001(v_HAL_61851_Listener);
      }
      // SECC_CmSlacMatch Behavior
      if (verdict == pass ) {
          verdict := f_SECC_CMN_TB_VTB_CmSlacMatch_001(inconc);
      }
      return verdict;
   }

   function f_SECC_CMN_PR_CmSetKey_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                    runs on SECC_Tester return verdicttype {

      var verdicttype verdict := f_SECC_CMN_PR_CmSlacMatch_001(v_HAL_61851_Listener);

      // SECC_CmSetKey Behavior
      if ( verdict == pass ) {
          tc_TT_match_join.start(par_TT_match_join);
          verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(false);
      }
      return verdict;
   }

   function f_SECC_CMN_PR_PLCLinkStatus_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                    runs on SECC_Tester return verdicttype {

        var verdicttype verdict := f_SECC_CMN_PR_CmSetKey_001(v_HAL_61851_Listener);

        // SECC_PLCLinkStatus Behavior
        if ( verdict == pass ) {
            verdict := f_SECC_CMN_TB_VTB_PLCLinkStatus_001(inconc);
            tc_TT_matching_repetition.stop;
        }
        return verdict;
   }

   function f_SECC_CMN_PR_CmAmpMap_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                 runs on SECC_Tester return verdicttype {

       var verdicttype verdict := f_SECC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);
       // EVCC_CmAmpMap Behavior
       if ( verdict == pass ) {
           if(PICS_CMN_CMN_InitiateCmAmpMap) {
               verdict := f_SECC_CMN_TB_VTB_CmAmpMap_001(inconc);
               if(getverdict == pass) {
                  f_SECC_CMN_checkTXPowerLimitation();
               }
           }
           else {
               verdict := f_SECC_CMN_TB_VTB_CmAmpMap_002(inconc);
           }
       }
       return verdict;
   }
}
```

D.2.2 EVCC + PLC bridge functions

```
module PreConditions_EVCC_15118_3 {

    import from TestBehavior_EVCC_CmSetKey all;
    import from TestBehavior_EVCC_CmSlacParm all;
    import from TestBehavior_EVCC_PLCLinkStatus all;
    import from TestBehavior_EVCC_AttenuationCharacterization all;
    import from TestBehavior_EVCC_CmSlacMatch all;
    import from TestBehavior_EVCC_CmValidate all;
    import from TestBehavior_EVCC_CmAmpMap all;
    import from TestBehavior_EVCC_CmValidateOrCmSlacMatch all;
    import from Timer_15118_3 all;
    import from ComponentsAndPorts all;
    import from LibFunctions_15118_3 { group generalFunctions; }
    import from Services_HAL_61851 all;
    import from Pics_15118_3 all;
    import from Pics_15118 all;
    import from Pixit_15118_3 all;
    import from Services_PLCLinkStatus all;

    import from Pixit_15118 all;
    import from Timer_15118 all;
    import from Services_TXPowerLimitation all;
    import from TTlibrary_Logging all;
    import from LibFunctions_15118_2 {function f_CMN_PhysicalValue_GetValue};

    function f_EVCC_CMN_PR_SetPowerFlowConfiguration_001() runs on EVCC_Tester return verdicttype{

        var verdicttype    verdict := pass;

        return verdict;
    }

    function f_EVCC_CMN_PR_DisableControlPilot_001() runs on EVCC_Tester return verdicttype {

        var verdicttype verdict := f_EVCC_CMN_PR_SetPowerFlowConfiguration_001();

            verdict := f_EVCC_setPwmMode(vc_errorState);
            sleep((par_CMN_waitForNextHAL));
            verdict := f_EVCC_setProximity(0);
        return verdict;
    }

   function f_EVCC_CMN_PR_CmSetKey_001() runs on EVCC_Tester return verdicttype {

        var verdicttype verdict := f_EVCC_CMN_PR_DisableControlPilot_001();

        // generate Nid and Nmk
        if ( verdict == pass ) {
            vc_Nmk := f_randomHexStringGen(32);
            vc_Nid := fx_generateNID(vc_Nmk);
            verdict := f_EVCC_CMN_TB_VTB_CmSetKey_001();
            f_EVCC_PLCNodeIsReadyForCommunication();
            sleep(par_EVCC_PLCNodeReady_delay);
        }
        return verdict;
    }

    function f_EVCC_CMN_PR_SetProximityPilot_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                             runs on EVCC_Tester return verdicttype {

        var verdicttype verdict := f_EVCC_CMN_PR_CmSetKey_001();

            if ( verdict == pass ) {
                v_HAL_61851_Listener.stop;
                v_HAL_61851_Listener.start(f_EVCC_HAL61851Listener(true));

                if(PICS_CMN_CMN_PlugType == type1) {
                    verdict := f_EVCC_setProximity(cc_proximity_type1);
                } else {
                    if(PICS_CMN_CMN_ChargingMode == aC){
                        var integer v_proximity_type2_AC;
                        if(PICS_CMN_AC_CableCapability == capability13A) {
                            v_proximity_type2_AC := cc_proximity_type2_AC_13A;
                        } else if(PICS_CMN_AC_CableCapability == capability20A) {
                            v_proximity_type2_AC := cc_proximity_type2_AC_20A;
                        } else if(PICS_CMN_AC_CableCapability == capability32A) {
                            v_proximity_type2_AC := cc_proximity_type2_AC_32A;
                        } else {
                            v_proximity_type2_AC := cc_proximity_type2_AC_63A;
                    }
                    verdict := f_EVCC_setProximity(v_proximity_type2_AC);
                } else {
                    verdict := f_EVCC_setProximity(cc_proximity_type2_DC);
                }
            }
        }
    return verdict;
}

function f_EVCC_CMN_PR_EnableControlPilot_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                              runs on EVCC_Tester return verdicttype {

    var verdicttype verdict := f_EVCC_CMN_PR_SetProximityPilot_001(v_HAL_61851_Listener);

        if ( verdict == pass ) {
            sleep((par_CMN_waitForNextHAL));
            f_EVCC_changeValidStateCondition(E,valid);
            verdict := f_EVCC_setPwmMode(e_PosVolt12);
        }
    return verdict;
}

function f_EVCC_CMN_PR_StateB_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                  runs on EVCC_Tester return verdicttype {

  var verdicttype verdict := f_EVCC_CMN_PR_EnableControlPilot_001(v_HAL_61851_Listener);

      if ( verdict == pass ) {
         timer statetimer := par_CMN_HAL_Timeout;
         verdict := f_EVCC_confirmState(valid, v_HAL_61851_Listener,
                                        statetimer, inconc);
       }
  tc_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min);

  return verdict;
}

function f_EVCC_CMN_PR_DutyCycle_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                     runs on EVCC_Tester return verdicttype {

    var integer v_oscOff := 100;
    var verdicttype verdict := f_EVCC_CMN_PR_StateB_001(v_HAL_61851_Listener);

        if ( verdict == pass ) {
            sleep((vc_DutyCycleDelay));
            if(PICS_CMN_CMN_ChargingMode == aC){
                if(vc_DutyCycle != v_oscOff) {
                    verdict := f_EVCC_setDutyCycle(vc_DutyCycle);
                    verdict := f_EVCC_setPwmMode(e_OscOn);
                }
            }
            else {
                verdict := f_EVCC_setDutyCycle(5);
                verdict := f_EVCC_setPwmMode(e_OscOn);
            }
            pt_SLAC_Port.clear;
        }
    return verdict;
}

function f_EVCC_CMN_PR_CmSlacParm_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                      runs on EVCC_Tester return verdicttype {

       var verdicttype verdict := f_EVCC_CMN_PR_DutyCycle_001(v_HAL_61851_Listener);
       if ( verdict == pass ) {
          verdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(inconc);
       }
       return verdict;
   }

function f_EVCC_CMN_PR_AttenuationCharacterization_001(out HAL_61851_Listener
                                                       v_HAL_61851_Listener)
                                                       runs on EVCC_Tester
                                                       return verdicttype {

   var verdicttype verdict := f_EVCC_CMN_PR_CmSlacParm_001(v_HAL_61851_Listener);
   if ( verdict == pass ) {
```

```
            verdict := f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(inconc);
        }
        return verdict;
    }

    function f_EVCC_CMN_PR_CmValidate_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                        runs on EVCC_Tester return verdicttype {

        var verdicttype verdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(
                                   v_HAL_61851_Listener);
        if ( verdict == passand PIXIT_EVCC_CMN_CmValidate == cmValidate) {
            verdict := f_EVCC_CMN_TB_VTB_CmValidate_001(v_HAL_61851_Listener, false,
                                                         vc_DutyCycle, inconc);
        }
        return verdict;
    }

    function f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                                     runs on EVCC_Tester return verdicttype {

        var verdicttype verdict := f_EVCC_CMN_PR_AttenuationCharacterization_001(
                                   v_HAL_61851_Listener);
        if ( verdict == pass) {
            verdict := f_EVCC_CMN_TB_VTB_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener,
                                                                      inconc);
        }
        return verdict;
    }

    function f_EVCC_CMN_PR_PLCLinkStatus_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                           runs on EVCC_Tester return verdicttype {

         var verdicttype verdict := f_EVCC_CMN_PR_CmValidateOrCmSlacMatch_001(v_HAL_61851_Listener);
         if ( verdict == pass ) {
             verdict := f_EVCC_CMN_TB_VTB_PLCLinkStatus_001(inconc);
         }
         return verdict;
    }

    function f_EVCC_CMN_PR_CmAmpMap_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                      runs on EVCC_Tester return verdicttype {

         var verdicttype verdict := f_EVCC_CMN_PR_PLCLinkStatus_001(v_HAL_61851_Listener);

         if ( verdict == pass ) {
             if(PICS_CMN_CMN_InitiateCmAmpMap) {
                 verdict := f_EVCC_CMN_TB_VTB_CmAmpMap_001(inconc);
                 if(getverdict == pass) {
                    f_EVCC_CMN_checkTXPowerLimitation();
                 }
             }
             else {
                 verdict := f_EVCC_CMN_TB_VTB_CmAmpMap_002(inconc);
             }
         }
         return verdict;
    }
}
```

D.3 Post-condition functions

D.3.1 SECC + PLC bridge functions

```
module PostConditions_SECC_15118_3 {

    import from ComponentsAndPorts all;
    import from Services_HAL_61851 all;
    import from LibFunctions_15118_3 { group generalFunctions; }
    import from Pics_15118 all;
    import from Pixit_15118 all;
    import from Timer_15118_3 all;
    import from Timer_15118 all;

    function f_SECC_CMN_PO_InitialState_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                          runs on SECC_Tester {

             v_HAL_61851_Listener.stop;
```

```
            v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(false));
            pt_HAL_61851_Port.clear;
            pt_HAL_61851_Internal_Port.clear;
            f_SECC_changeValidStateCondition(A);
            f_SECC_changeValidFrequencyRange(0,0);
            f_SECC_changeValidDutyCycleRange(100,100);
            f_SECC_setState(A,v_HAL_61851_Listener);
            pt_HAL_61851_Port.clear;
            sleep((par_CMN_waitForNextHAL));
            f_SECC_setProximity(0);
            v_HAL_61851_Listener.stop;


        all timer.stop;

        log(par_SECC_waitForNextTC, " Sec timer started");
        sleep(par_SECC_waitForNextTC);
        log(par_SECC_waitForNextTC, " Sec timer stopped");
     }
}
```

D.3.2 EVCC + PLC bridge functions

```
module PostConditions_EVCC_15118_3 {

    import from ComponentsAndPorts all;
    import from Services_HAL_61851 all;
    import from LibFunctions_15118_3 all;
    import from Pics_15118 all;
    import from Timer_15118_3 all;
    import from Pixit_15118 all;
    import from Timer_15118 all;

    function f_EVCC_CMN_PO_InitialState_001(out HAL_61851_Listener v_HAL_61851_Listener)
                                        runs on EVCC_Tester {
            v_HAL_61851_Listener.stop;
            v_HAL_61851_Listener.start(f_EVCC_HAL61851Listener(false));

            pt_HAL_61851_Port.clear;
            f_EVCC_setPwmMode(e_OscOff);
            pt_HAL_61851_Port.clear;
            sleep((par_CMN_waitForNextHAL));
            f_EVCC_setProximity(0);
            v_HAL_61851_Listener.stop;



        all timer.stop;

        log((par_EVCC_waitForNextTC)," Sec timer started");
        sleep((par_EVCC_waitForNextTC));
        log((par_EVCC_waitForNextTC)," Sec timer stopped");
     }
}
```

D.4 Library functions

```
module LibFunctions_15118_3 {

    group generalFunctions {

        type record ListOfFloat {
            record length (0 .. 10) of float v_float
        }

        type record ListOfBoolean {
            record length (0 .. 10) of boolean v_boolean
        }

        external function fx_logToFile(in template ListOfFloat v_listOfFloat,
                                       in template ListOfBoolean v_listOfBoolean);

        external function fx_generateNID(hexstring nmk) return hexstring;

        external function fx_captureTraffic(in integer v_interfaceIdx);
        external function fx_stopCapturing();

        function f_randomHexStringGen(integer hexLength) return hexstring {
            var hexstring randomHex := ''H;

            for (var integer i:=0; i<hexLength/2; i:=i + 1) {
                var float rndFloat := -1.0;

                while(rndFloat<0.0 or rndFloat>255.0){
                    rndFloat := rnd(rnd());
                    rndFloat := rndFloat*10E2;
                }
                var hexstring randomHexByte := int2hex(float2int(rndFloat),2);
                randomHex := randomHex & randomHexByte;
            }

            return randomHex;
        }

        function sleep(float time) {
            timer t := time;
            t.start;
            t.timeout;
        }
    }
}
```

Annex E
(normative)

Function specifications for 15118-3

E.1 SECC + PLC bridge functions

This subclause includes all function *specifications* where the EVSE is defined as SUT.

E.1.1 SECC functions for CmSlacParm

```
module TestBehavior_SECC_CmSlacParm {

    import from Timer_15118_3 all;
    import from Templates_CMN_CmSlacParm all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from DataStructure_HAL_61851 all;
    import from Services_HAL_61851 all;
    import from Pics_15118_3 all;
    import from Pics_15118 all;
    import from Pixit_15118_3 all;
    import from LibFunctions_15118_3  { group generalFunctions; }
    import from Services_PLCLinkStatus all;
    import from Timer_15118 all;

    function f_SECC_CMN_TB_VTB_CmSlacParm_001(in verdicttype v_vct)
                                          runs on SECC_Tester
                                          return verdicttype {

        var MME v_responseMessage;
        var boolean v_repetition := true;
        var integer v_count1 := 0;
        var integer v_count2 := 0;
        var MACAddress_TYPE v_sut_mac;

        vc_macAddresList := m_CMN_CMN_EmptyMacAddresList();
        vc_RunID := f_randomHexStringGen(16);

        while(v_repetition){

            tc_TT_match_response.start(par_TT_match_response);
            v_count1 := v_count1 + 1;
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(md_CMN_CMN_SlacMmeCmnHeader_001({
                                                  CM_SLAC_PARM_REQ := '6064'H}),
                                                  md_CMN_CMN_CmSlacParmReq_001(
                                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                                  to cc_eth_broadcast;

             alt {
                    []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_SLAC_PARM_CNF := '6065'H}),
                                      md_CMN_CMN_CmSlacParmCnf_001(par_testSystem_mac,
                                      m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                      -> value v_responseMessage sender v_sut_mac {

                        setverdict(pass,"CM_SLAC_PARM is correct.");
                        vc_macAddresList.macAddressList[v_count2] := v_sut_mac;
                        v_count2 := v_count2 + 1;
                        vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                         cm_slac_parm_cnf.num_sounds;
                        vc_Time_out := v_responseMessage.mme_payload.payload.
                                       cm_slac_parm_cnf.time_out;
                        repeat;
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
```

```
                    setverdict(v_vct, "Invalid message type or content was received.");
                    v_repetition := false;
                }
                [] tc_TT_match_response.timeout {
                     if(sizeof(vc_macAddresList.macAddressList) > 0){
                         tc_TP_match_sequence.start(par_TP_match_sequence);
                         v_repetition := false;
                     }
                     else if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                         log("The Matching process is considered as FAILED.");
                         if(tc_TT_matching_repetition.running){
                             log("TT_matching_repetition is still running. " &
                                 "A new Matching process is started.");
                             v_count1 := 0;
                         }
                         else {
                             setverdict(v_vct, "TT_matching_repetition has expired. " &
                                               "No new Matching process will be started.");
                             v_repetition := false;
                         }
                     }
                }
        }
    }
    return getverdict;
}

function f_SECC_CMN_TB_VTB_CmSlacParm_002(in HAL_61851_Listener v_HAL_61851_Listener,
                                          in template SLAC_Header v_slac_Header,
                                          in boolean v_sendInvalid)
                                          runs on SECC_Tester return verdicttype {

    var MME v_responseMessage;
    var boolean v_repetition := true;

    vc_RunID := f_randomHexStringGen(16);

    tc_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min);
    if(v_sendInvalid) {
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({CM_SLAC_PARM_REQ := '6064'H}),
                          md_CMN_CMN_CmSlacParmReq_001(v_slac_Header, vc_RunID)))
                          to cc_eth_broadcast;
    }

    alt {
        []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                               md_CMN_CMN_SlacMmeCmnHeader_001({
                               CM_SLAC_PARM_CNF := '6065'H}),
                               md_CMN_CMN_CmSlacParmCnf_001(par_testSystem_mac,
                               m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID))) {

            setverdict(fail,"Invalid CM_SLAC_PARM.REQ message was not ignored.");
            tc_TT_EVSE_SLAC_init.stop;
        }
        [] a_SECC_processPLCLinkNotifications_001();
        [] pt_SLAC_Port.receive {
           setverdict(fail, "Invalid message type or content was received.");
        }
        [] tc_TT_EVSE_SLAC_init.timeout {

            v_HAL_61851_Listener.stop;
            v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(false));
            f_SECC_changeValidStateCondition(valid_Matching);
            f_SECC_changeValidFrequencyRange(0,0);
            f_SECC_changeValidDutyCycleRange(100,100);
            tc_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_max - par_TT_EVSE_SLAC_init_min);
            alt {
                [] tc_TT_EVSE_SLAC_init.timeout;
            }

            tc_TT_match_response.start(par_TT_match_response);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_SLAC_PARM_REQ := '6064'H}),
                              md_CMN_CMN_CmSlacParmReq_001(
                              m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                              to cc_eth_broadcast;
            alt {
                    []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                             md_CMN_CMN_SlacMmeCmnHeader_001({
                                             CM_SLAC_PARM_CNF := '6065'H}),?)) {

                        setverdict(fail,"CM_SLAC_PARM.CNF was sent from the SUT although " &
                                        "the timer TT_EVSE_SLAC_init should have " &
                                        "been expired.");
                        tc_TT_match_response.stop;
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(fail, "Invalid message type or content was received.");
                    }
                    [] tc_TT_match_response.timeout {
                       setverdict(pass,"TT_match_response timeout. " &
                                      "CM_SLAC_PARM.CNF was not sent from the SUT " &
                                      "because the timer TT_EVSE_SLAC_init is expired.");
                    }
                }
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_CmSlacParm_003(in HAL_61851_Listener v_HAL_61851_Listener)
                                            runs on SECC_Tester return verdicttype {

        vc_RunID := f_randomHexStringGen(16);
        sleep(1.0);

             f_SECC_changeValidStateCondition(invalid);
             f_SECC_changeValidFrequencyRange(0,0);
             f_SECC_changeValidDutyCycleRange(100,100);
             deactivate(vc_Default_IEC_61851_ListenerBehavior);
             f_SECC_setState(A,v_HAL_61851_Listener);


        tc_TT_match_response.start(par_TT_match_response);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(md_CMN_CMN_SlacMmeCmnHeader_001({
                                                 CM_SLAC_PARM_REQ := '6064'H}),
                                                 md_CMN_CMN_CmSlacParmReq_001(
                                                 m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                                 to cc_eth_broadcast;

        alt {
            []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_SLAC_PARM_CNF := '6065'H}),
                                   md_CMN_CMN_CmSlacParmCnf_001(par_testSystem_mac,
                                   m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID))) {

                setverdict(fail,"CM_SLAC_PARM.CNF message was not expected." &
                                "CP State A should be detected before.");
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_match_response.timeout {
                setverdict(pass,"TT_match_response timer has expired, " &
                                "the Matching process was terminated by the SUT.");
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_CmSlacParm_004() runs on SECC_Tester return verdicttype {

        var MME v_responseMessage;
        var boolean v_repetition := true;
        var integer v_count1 := 0;
        var integer v_count2 := 0;
        var MACAddress_TYPE v_sut_mac;

        tc_T_conn_max_comm.start;

        vc_macAddresList := m_CMN_CMN_EmptyMacAddresList();
        vc_RunID := f_randomHexStringGen(16);
    while(v_repetition){

        tc_TT_match_response.start(par_TT_match_response);
        v_count1 := v_count1 + 1;
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_SLAC_PARM_REQ := '6064'H}),
                                          md_CMN_CMN_CmSlacParmReq_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                          to cc_eth_broadcast;

         alt {
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_SLAC_PARM_CNF := '6065'H}),
                                    md_CMN_CMN_CmSlacParmCnf_001(par_testSystem_mac,
                                    m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                    -> value v_responseMessage sender v_sut_mac {

                    setverdict(pass,"CM_SLAC_PARM is correct.");
                    vc_macAddresList.macAddressList[v_count2] := v_sut_mac;
                    v_count2 := v_count2 + 1;
                    vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                     cm_slac_parm_cnf.num_sounds;
                    vc_Time_out := v_responseMessage.mme_payload.payload.
                                   cm_slac_parm_cnf.time_out;
                    repeat;
                }
                [] a_SECC_processPLCLinkNotifications_001();
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                   v_repetition := false;
                }
                [] tc_TT_match_response.timeout {
                    if(sizeof(vc_macAddresList.macAddressList) > 0){
                        tc_TP_match_sequence.start(par_TP_match_sequence);
                        v_repetition := false;
                    }
                    else if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                        log("The Matching process is considered as FAILED.");
                        if(tc_TT_matching_repetition.running){
                            log("TT_matching_repetition is still running. " &
                                "A new Matching process is started.");
                            v_count1 := 0;
                        }
                        else {
                            setverdict(fail, "TT_matching_repetition has expired. " &
                                             "No new Matching process will be started.");
                            v_repetition := false;
                        }
                    }
                }
                [] tc_T_conn_max_comm.timeout {
                    setverdict(fail, "T_conn_max_comm has expired. " &
                                     "The SUT was not ready for communication " &
                                     "within 'T_conn_max_comm' after wakeup by plug-in");
                }
         }
    }
    return getverdict;
}

function f_SECC_CMN_TB_VTB_CmSlacParm_005(in HAL_61851_Listener v_HAL_61851_Listener)
                                          runs on SECC_Tester return verdicttype {

    var MME v_responseMessage;
    var MACAddress_TYPE v_sut_mac;

    f_SECC_setIsConfirmationFlagDC();
    f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                            par_T_conn_max_comm,
                            fail);

    vc_macAddresList := m_CMN_CMN_EmptyMacAddresList();
    vc_RunID := f_randomHexStringGen(16);

    tc_TT_match_response.start(par_TT_match_response);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_SLAC_PARM_REQ := '6064'H}),
                                            md_CMN_CMN_CmSlacParmReq_001(
                                            m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                            to cc_eth_broadcast;

     alt {
            []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_PARM_CNF := '6065'H}),
                                 md_CMN_CMN_CmSlacParmCnf_001(par_testSystem_mac,
                                 m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                 -> value v_responseMessage sender v_sut_mac {

                setverdict(pass,"CM_SLAC_PARM is correct.");
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_match_response.timeout {
                setverdict(fail,"TT_match_response timeout. " &
                                "The SUT did not respond to CM_SLAC_PARM.REQ message " &
                                "after 5% duty cycle detection.");
        }
    }
    return getverdict;
}

function f_SECC_AC_TB_VTB_CmSlacParm_001(in HAL_61851_Listener v_HAL_61851_Listener)
                                         runs on SECC_Tester return verdicttype {

    if(not vc_confirmState) {
        timer statetimer := par_CMN_HAL_Timeout;
        f_SECC_confirmState(EorF, v_HAL_61851_Listener, statetimer);
        if(getverdict == pass) {
            tc_T_step_EF.start(par_T_step_EF_min - cc_offset);
        }
    }

    if(getverdict != pass) {
       setverdict(fail, "The SUT did not apply CP State E or F.");
    } else {
        alt {
            [] tc_T_step_EF.timeout;
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
        }

        f_SECC_changeValidStateCondition(B);
        f_SECC_changeValidFrequencyRange(980,1020);
        f_SECC_changeValidDutyCycleRange(10,96);
        vc_validDutyCycleLowerBound1 := 10;
        vc_validDutyCycleUpperBound1 := 96;
        vc_validDutyCycleLowerBound2 := 10;
        vc_validDutyCycleUpperBound2 := 96;
        f_SECC_setIsConfirmationFlagDC();

        tc_T_step_EF.start(par_T_step_EF_max -
                           (par_T_step_EF_min - cc_offset));
        alt {
            [] tc_T_step_EF.timeout;
            [] a_SECC_DCDetection(pt_HAL_61851_Internal_Port,
                                 vc_validDutyCycleLowerBound1,
                                 vc_validDutyCycleUpperBound1,
                                 vc_validDutyCycleLowerBound2,
                                 vc_validDutyCycleUpperBound2) {
               vc_confirmDC := true;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
        }

        if(not vc_confirmDC) {
            f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                    par_T_conn_max_comm,
                                fail);
            }

            if(getverdict == pass) {
                setverdict(pass, "The EVSE could signal a nominal duty cycle.");
            }
        }
        return getverdict;
    }

    function f_SECC_AC_TB_VTB_CmSlacParm_002(in HAL_61851_Listener v_HAL_61851_Listener)
                                          runs on SECC_Tester return verdicttype {

        var integer v_count := 1;
        tc_TP_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min - cc_offset);

        alt {
            [] tc_TP_EVSE_SLAC_init.timeout;
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
        }

        f_SECC_changeValidStateCondition(EorF);
        f_SECC_changeValidFrequencyRange(0,0);
        f_SECC_changeValidDutyCycleRange(0,0);
        f_SECC_setIsConfirmationFlagVoltage();

        tc_TP_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_max -
                                  (par_TT_EVSE_SLAC_init_min - cc_offset));
        alt {
            [] tc_TP_EVSE_SLAC_init.timeout;
            [] a_SECC_EFDetection(pt_HAL_61851_Internal_Port, EorF) {
               vc_confirmState := true;
               tc_T_step_EF.start(par_T_step_EF_min - cc_offset);
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
        }

        while ((v_count <= par_C_sequ_retry) and (getverdict == pass)) {

            if(not vc_confirmState) {
                timer statetimer := par_CMN_HAL_Timeout;
                f_SECC_confirmState(EorF, v_HAL_61851_Listener, statetimer);
                tc_T_step_EF.start(par_T_step_EF_min - cc_offset);
            }
            vc_confirmState := false;
            if(getverdict == pass) {
                alt {
                    [] tc_T_step_EF.timeout;
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(fail, "Invalid message type or content was received.");
                    }
                }

                f_SECC_changeValidStateCondition(B);
                f_SECC_changeValidFrequencyRange(980,1020);
                f_SECC_changeValidDutyCycleRange(3,7);

                tc_T_step_EF.start(par_T_step_EF_max -
                                  (par_T_step_EF_min - cc_offset));
                alt {
                    [] tc_T_step_EF.timeout;
                    [] a_SECC_EFDetection(pt_HAL_61851_Internal_Port, EorF) {
                       vc_confirmState := true;
                       tc_TP_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min - cc_offset);
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(fail, "Invalid message type or content was received.");
                    }
                }
                if(not vc_confirmState) {
                    f_SECC_setIsConfirmationFlagDC();
                f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                        par_T_conn_max_comm,
                                        fail);
                tc_TP_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min - cc_offset);
            }
            vc_confirmState := false;
            v_count := v_count + 1;
            alt {
                [] tc_TP_EVSE_SLAC_init.timeout;
                [] a_SECC_processPLCLinkNotifications_001();
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                }
            }

            f_SECC_changeValidFrequencyRange(0,0);
            if(v_count == par_C_sequ_retry) {
                f_SECC_changeValidDutyCycleRange(100,100);
                vc_validDutyCycleLowerBound1 := 100;
                vc_validDutyCycleUpperBound1 := 100;
                vc_validDutyCycleLowerBound2 := 100;
                vc_validDutyCycleUpperBound2 := 100;
                f_SECC_setIsConfirmationFlagDC();
            }
            else {
                f_SECC_changeValidDutyCycleRange(0,0);
                f_SECC_changeValidStateCondition(EorF);
                f_SECC_setIsConfirmationFlagVoltage();
            }

            tc_TP_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_max -
                                      (par_TT_EVSE_SLAC_init_min - cc_offset));

            alt {
                [] tc_TP_EVSE_SLAC_init.timeout;
                [] a_SECC_EFDetection(pt_HAL_61851_Internal_Port, EorF) {
                   vc_confirmState := true;
                   tc_T_step_EF.start(par_T_step_EF_min - cc_offset);
                }
                [] a_SECC_processPLCLinkNotifications_001();
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                }
            }
        }
    }
    if(not vc_confirmState) {
        f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                par_T_conn_max_comm,
                                fail);
    }

    if(getverdict == pass) {
        setverdict(pass, "The SUT has initiated an oscillator shutdown " &
                         "after the repetition of 2 fallback sequences (5%).");
    }
    return getverdict;
}

function f_SECC_AC_TB_VTB_CmSlacParm_003(in HAL_61851_Listener v_HAL_61851_Listener)
                                         runs on SECC_Tester return verdicttype {

    f_SECC_changeValidStateCondition(valid_Matching);
    f_SECC_changeValidDutyCycleRange(10,96);
    vc_validDutyCycleLowerBound1 := 10;
    vc_validDutyCycleUpperBound1 := 96;
    vc_validDutyCycleLowerBound2 := 10;
    vc_validDutyCycleUpperBound2 := 96;

    f_SECC_setState(B,v_HAL_61851_Listener);
    tc_TT_matching_repetition.start(par_TT_matching_repetition);

    f_SECC_setIsConfirmationFlagDC();
    f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                            par_T_conn_max_comm,
                            fail);
    if(getverdict != pass) {
        setverdict(fail, "No nominal duty cycle could be detected.");
        }

        return getverdict;
    }
 }
```

E.1.2 SECC functions for AttenuationCharacterization

```
module TestBehavior_SECC_AttenuationCharacterization {

    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Pics_15118 all;
    import from Templates_CMN_CmSlacParm all;
    import from Templates_SECC_CmAttenCharInd all;
    import from Templates_CMN_CmAttenCharRsp all;
    import from Templates_CMN_CmStartAttenCharInd all;
    import from Templates_CMN_CmMnbcSoundInd all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmSlacMatch all;
    import from ComponentsAndPorts all;
    import from Services_HAL_61851 all;
    import from DataStructure_SLAC all;
    import from LibFunctions_15118_3  { group generalFunctions; }
    import from Services_PLCLinkStatus all;
    import from Timer_15118 all;
    import from Services_TXPowerLimitation all;
    import from Services_HAL_61851 all;

    function f_SECC_CMN_TB_VTB_AttenuationCharacterization_001(in verdicttype v_vct)
                                                            runs on SECC_Tester
                                                            return verdicttype {

           var MME v_responseMessage;
           var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
           vc_LowestAverageAttenuation := 0.0;
           var MACAddress_TYPE v_sut_mac;

           tc_TP_match_sequence.timeout;
           tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

           for (var integer i:=0; i<3; i:=i+1) {
               tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
               pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_START_ATTEN_CHAR_IND := '606A'H}),
                                  md_CMN_CMN_CmStartAttenCharInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                                  vc_Time_out, '01'H, par_testSystem_mac, vc_RunID)))
                                  to cc_eth_broadcast;

               tc_TP_EV_batch_msg_interval.timeout;
           };

           var integer v_cnt := par_C_EV_match_MNBC;
           for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
               v_cnt := v_cnt -1;
               tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
               pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_MNBC_SOUND_IND := '6076'H}),
                                  md_CMN_CMN_CmMnbcSoundInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), int2hex(v_cnt,2),
                                  vc_RunID, v_source_rnd))) to cc_eth_broadcast;

               tc_TP_EV_batch_msg_interval.timeout;

           };

           var integer v_cnt_pot_evse := 0;

           alt {
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_ATTEN_CHAR_IND := '606E'H}),
                                       mdw_SECC_CMN_CmAttenCharInd_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(),
                            par_testSystem_mac, vc_RunID, ?)))
                            -> value v_responseMessage sender v_sut_mac {

             setverdict(pass,"CM_ATTEN_CHAR.IND is correct.");
             v_cnt_pot_evse := v_cnt_pot_evse + 1;
             vc_attenuation_list := v_responseMessage.mme_payload.payload.
                                    cm_atten_char_ind.attenuation_list;
             tc_TP_match_sequence.start(par_TP_match_sequence);

             pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_ATTEN_CHAR_RSP :='606F'H}),
                              md_CMN_CMN_CmAttenCharRsp_001(
                              m_CMN_CMN_SlacPayloadHeader_001(),
                              md_CMN_CMN_Acvarfield_001(par_testSystem_mac, vc_RunID))))
                              to v_sut_mac;

             tc_TP_match_sequence.stop;
             f_SECC_CMN_setMac(v_responseMessage, v_sut_mac);
             if(sizeof(vc_macAddresList.macAddressList) == v_cnt_pot_evse) {
              log("CM_ATTEN_CHAR.IND messages from all EVSEs are received.");
             }
             else{repeat;}
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SLAC_PARM_CNF := '6065'H}),?)) {
            // CM_SLAC_PARM.CNF messages will be ignored!
            repeat;
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
            // CM_ATTEN_PROFILE.IND messages will be ignored!
            repeat;
         }
         [] a_SECC_processPLCLinkNotifications_001();
         [] pt_SLAC_Port.receive {
            setverdict(v_vct, "Invalid message type or content was received.");
         }
         [] tc_TT_EV_atten_results.timeout {
            if(v_cnt_pot_evse == 0){
                 setverdict(v_vct,"TT_EV_atten_results timeout and " &
                                    "no CM_ATTEN_CHAR.IND " &
                                    "received - EVSE_NOT_FOUND.");
            }
         }
         [] tc_TT_matching_repetition.timeout {
            log("TT_matching_repetition timeout - " &
                  "No new matching process can be started, " &
                "if the current matching process fails.");
            repeat;
         }
      }
      return getverdict;
  }

function f_SECC_CMN_TB_VTB_AttenuationCharacterization_002() runs on SECC_Tester
                                                             return verdicttype {

       var MME v_responseMessage;
       var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
       vc_LowestAverageAttenuation := 0.0;

       tc_TP_match_sequence.timeout;
       tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

       for (var integer i:=0; i<3; i:=i+1) {
           tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_START_ATTEN_CHAR_IND := '606A'H}),
                              md_CMN_CMN_CmStartAttenCharInd_001(
                              m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                              vc_Time_out, '01'H, par_testSystem_mac, vc_RunID)))
                              to cc_eth_broadcast;

           tc_TP_EV_batch_msg_interval.timeout;
```

```
        };

        var integer v_cnt := par_C_EV_match_MNBC;
        for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
            v_cnt := v_cnt -1;
            tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_MNBC_SOUND_IND := '6076'H}),
                                md_CMN_CMN_CmMnbcSoundInd_001(
                                m_CMN_CMN_SlacPayloadHeader_001(), int2hex(v_cnt,2),
                                vc_RunID, v_source_rnd))) to cc_eth_broadcast;

            tc_TP_EV_batch_msg_interval.timeout;

        };

        alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_ATTEN_CHAR_IND := '606E'H}),
                                    mdw_SECC_CMN_CmAttenCharInd_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    par_testSystem_mac, vc_RunID, ?)))
                                    -> value v_responseMessage {

                setverdict(pass,"CM_ATTEN_CHAR.IND is correct.");
                tc_TT_EV_atten_results.stop;

                vc_RunID := f_randomHexStringGen(16);
                tc_TP_match_sequence.start(par_TP_match_sequence);

                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_SLAC_PARM_REQ := '6064'H}),
                                  md_CMN_CMN_CmSlacParmReq_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                  to cc_eth_broadcast;
                alt {
                    []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_SLAC_PARM_CNF := '6065'H}),
                                           md_CMN_CMN_CmSlacParmCnf_001(
                                           par_testSystem_mac ,
                                           m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID))) {

                        setverdict(pass,"CM_SLAC_PARM.CNF is correct.");
                        tc_TP_match_sequence.stop;
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(fail, "Invalid message type or content was received.");
                    }
                    [] tc_TT_match_response.timeout {
                       setverdict(fail,"TT_match_response timeout. " &
                                         "The SECC did not reply to CM_SLAC_PARM " &
                                        "request message.");
                    }
                    [] tc_TT_matching_repetition.timeout {
                       log("TT_matching_repetition timeout - " &
                             "No new matching process can be started, " &
                           "if the current matching process fails.");
                       repeat;
                    }
                 }
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_SLAC_PARM_CNF := '6065'H}),?)) {
               // CM_SLAC_PARM.CNF messages will be ignored!
               repeat;
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_ATTEN_PROFILE_IND := '6086'H}),?)) {

               // CM_ATTEN_PROFILE.IND messages will be ignored!
               repeat;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_EV_atten_results.timeout {
               setverdict(fail,"TT_EV_atten_results timeout and no CM_ATTEN_CHAR.IND " &
                                  "received - EVSE_NOT_FOUND.");
            }
            [] tc_TT_matching_repetition.timeout {
               log("TT_matching_repetition timeout - " &
                     "No new matching process can be started, " &
                   "if the current matching process fails.");
               repeat;
            }
         }
         return getverdict;
     }

function f_SECC_CMN_TB_VTB_AttenuationCharacterization_003(in integer n) runs on SECC_Tester
                                                                           return verdicttype {

        var MME v_responseMessage;
        var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);

        tc_TP_match_sequence.timeout;
        tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

        for (var integer i:=0; i<3; i:=i+1) {
            tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_START_ATTEN_CHAR_IND := '606A'H}),
                              md_CMN_CMN_CmStartAttenCharInd_001(
                              m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                              vc_Time_out, '01'H, par_testSystem_mac, vc_RunID)))
                              to cc_eth_broadcast;

            tc_TP_EV_batch_msg_interval.timeout;
        };

        var integer v_cnt := par_C_EV_match_MNBC;
        // send (par_C_EV_match_MNBC-n) CM_MNBC_SOUND.IND messages
        for (var integer i:=0; i<par_C_EV_match_MNBC-n; i:=i+1) {
            v_cnt := v_cnt -1;
            tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_MNBC_SOUND_IND := '6076'H}),
                              md_CMN_CMN_CmMnbcSoundInd_001(
                              m_CMN_CMN_SlacPayloadHeader_001(),
                              int2hex(v_cnt,2), vc_RunID, v_source_rnd)))
                              to cc_eth_broadcast;

            tc_TP_EV_batch_msg_interval.timeout;

        };

        alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_ATTEN_CHAR_IND := '606E'H}),
                                    mdw_SECC_CMN_CmAttenCharInd_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    par_testSystem_mac, vc_RunID, ?)))
                                    -> value v_responseMessage {

               setverdict(pass,"Anticipated number of CM_MNBC_SOUND.IND " &
                                  "messages was not sent, " &
                                  "but CM_ATTEN_CHAR.IND is correct.");
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_SLAC_PARM_CNF := '6065'H}),?)) {
               // CM_SLAC_PARM.CNF messages will be ignored!
               repeat;
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
```

```
                                 CM_ATTEN_PROFILE_IND := '6086'H}),?)) {

               // CM_ATTEN_PROFILE.IND messages will be ignored!
               repeat;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_EV_atten_results.timeout {
               setverdict(fail,"TT_EV_atten_results timeout and no CM_ATTEN_CHAR.IND " &
                                 "received - EVSE_NOT_FOUND.");
            }
            [] tc_TT_matching_repetition.timeout {
               log("TT_matching_repetition timeout - " &
                     "No new matching process can be started, " &
                   "if the current matching process fails.");
               repeat;
            }
         }
         return getverdict;
      }

   function f_SECC_CMN_TB_VTB_AttenuationCharacterization_004() runs on SECC_Tester
                                                                  return verdicttype {

          var MME v_responseMessage;
          var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
          vc_LowestAverageAttenuation := 0.0;
          var integer v_count := 0;
          var MACAddress_TYPE v_sut_mac;

          tc_TP_match_sequence.timeout;
          tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

          for (var integer i:=0; i<3; i:=i+1) {
              tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
              pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_START_ATTEN_CHAR_IND := '606A'H}),
                                  md_CMN_CMN_CmStartAttenCharInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                                  vc_Time_out, '01'H, par_testSystem_mac, vc_RunID)))
                                  to cc_eth_broadcast;

              tc_TP_EV_batch_msg_interval.timeout;
          };

          var integer v_cnt := par_C_EV_match_MNBC;
          for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
              v_cnt := v_cnt -1;
              tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
              pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_MNBC_SOUND_IND := '6076'H}),
                                  md_CMN_CMN_CmMnbcSoundInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(),
                                  int2hex(v_cnt,2), vc_RunID, v_source_rnd)))
                                  to cc_eth_broadcast;

              tc_TP_EV_batch_msg_interval.timeout;

          };

          alt {
              [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_ATTEN_CHAR_IND := '606E'H}),
                                      mdw_SECC_CMN_CmAttenCharInd_001(
                                      m_CMN_CMN_SlacPayloadHeader_001(),
                                      par_testSystem_mac, vc_RunID, ?)))
                                      -> sender v_sut_mac {

                  if(v_count > 0){
                      setverdict(pass,"CM_ATTEN_CHAR.IND message was repeated.",v_count);
                  }
                  v_count := v_count + 1;
                  tc_TT_match_response.start(par_TT_match_response +
                                              par_CMN_Transmission_Delay);
```

```
    if(v_count > par_C_EV_match_retry) {
      alt{
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_IND := '606E'H}),
                                 mdw_SECC_CMN_CmAttenCharInd_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(),
                                 par_testSystem_mac, vc_RunID, ?)))
                                 -> sender v_sut_mac {

          setverdict(fail,"CM_ATTEN_CHAR.IND message was " &
                           "repeated, but v_count > " &
                          "par_C_EV_match_retry.");
        }
        [] tc_TT_match_response.timeout {
          setverdict(pass,"TT_match_response timeout. " &
                          "The total number of retries is reached, " &
                          "the Matching process " &
                          "shall be considered as FAILED");

          pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_ATTEN_CHAR_RSP :='606F'H}),
                            md_CMN_CMN_CmAttenCharRsp_001(
                            m_CMN_CMN_SlacPayloadHeader_001(),
                            md_CMN_CMN_Acvarfield_001(
                            par_testSystem_mac, vc_RunID))))
                            to v_sut_mac;

          tc_TT_match_response.start(par_TT_match_response);
          pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_SLAC_MATCH_REQ := '607C'H}),
                            md_CMN_CMN_CmSlacMatchReq_001(
                            m_CMN_CMN_SlacPayloadHeader_001(),
                            par_testSystem_mac, v_sut_mac, vc_RunID)))
                            to v_sut_mac;

          alt {
                 []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_SLAC_MATCH_CNF := '607D'H}),
                                        md_CMN_CMN_CmSlacMatchCnf_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        par_testSystem_mac, v_sut_mac,
                                        vc_RunID, ?, ?))) {

                     setverdict(fail,"CM_SLAC_MATCH.CNF message " &
                                     "was not expected. " &
                                     "Repetition limit was reached.");
                     tc_TT_match_response.stop;
                 }
                 [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                     // CM_ATTEN_CHAR.IND messages will be ignored!
                     repeat;
                 }
                 [] a_SECC_processPLCLinkNotifications_001();
                 [] pt_SLAC_Port.receive {
                    setverdict(fail, "Invalid message type or content " &
                                    "was received.");
                 }
                 [] tc_TT_match_response.timeout {
                     setverdict(pass,"TT_match_response timeout. " &
                                     "Matching process is " &
                                     "considered as FAILED.");
                 }
                 [] tc_TT_matching_repetition.timeout {
                    log("TT_matching_repetition timeout - " &
                        "No new matching process can be started, " &
                        "if the current matching process fails.");
                    repeat;
                }
            }
        }
```

```
                }
              }
              else{
                  repeat;
              }
          }
          [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_PARM_CNF := '6065'H}),?)) {
             // CM_SLAC_PARM.CNF messages will be ignored!
             repeat;
          }
          [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_ATTEN_PROFILE_IND := '6086'H}),?)) {

             // CM_ATTEN_PROFILE.IND messages will be ignored!
             repeat;
          }
          [] a_SECC_processPLCLinkNotifications_001();
          [] pt_SLAC_Port.receive {
             setverdict(fail, "Invalid message type or content was received.");
          }
          [] tc_TT_EV_atten_results.timeout {
             setverdict(fail,"TT_EV_atten_results timeout and no CM_ATTEN_CHAR.IND " &
                              "received - EVSE_NOT_FOUND.");
          }
          [] tc_TT_match_response.timeout {
             setverdict(fail,"TT_match_response timeout. " &
                              "CM_ATTEN_CHAR.IND message was not repeated.");
          }
          [] tc_TT_matching_repetition.timeout {
             log("TT_matching_repetition timeout - " &
                   "No new matching process can be started, " &
                 "if the current matching process fails.");
             repeat;
          }
       }
       return getverdict;
    }

function f_SECC_CMN_TB_VTB_AttenuationCharacterization_005(in template(present)
                                                           SLAC_Header payloadHeader,
                                                           in template(present)
                                                           Acvarfield_Type acvarfield)
                                                           runs on SECC_Tester
                                                           return verdicttype {

       var MME v_responseMessage;
       var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
       vc_LowestAverageAttenuation := 0.0;
       var integer v_count := 0;
       var MACAddress_TYPE v_sut_mac;

       tc_TP_match_sequence.timeout;
       tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

       for (var integer i:=0; i<3; i:=i+1) {
           tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_START_ATTEN_CHAR_IND := '606A'H}),
                             md_CMN_CMN_CmStartAttenCharInd_001(
                             m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds, vc_Time_out,
                             '01'H, par_testSystem_mac, vc_RunID))) to cc_eth_broadcast;

           tc_TP_EV_batch_msg_interval.timeout;
       };

       var integer v_cnt := par_C_EV_match_MNBC;
       for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
           v_cnt := v_cnt -1;
           tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_MNBC_SOUND_IND := '6076'H}),
                             md_CMN_CMN_CmMnbcSoundInd_001(
                             m_CMN_CMN_SlacPayloadHeader_001(), int2hex(v_cnt,2),
                             vc_RunID, v_source_rnd))) to cc_eth_broadcast;
```

```
        tc_TP_EV_batch_msg_interval.timeout;

    };

    alt {
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_ATTEN_CHAR_IND := '606E'H}),
                                mdw_SECC_CMN_CmAttenCharInd_001(
                                m_CMN_CMN_SlacPayloadHeader_001(),
                                par_testSystem_mac, vc_RunID, ?)))
                                -> sender v_sut_mac {

            if(v_count > 0){
                setverdict(pass,"CM_ATTEN_CHAR.IND message was repeated.",v_count);
            }
            v_count := v_count + 1;
            tc_TT_match_response.start(par_TT_match_response +
                                       par_CMN_Transmission_Delay);

            // send invalid CM_ATTEN_CHAR.RSP message
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_ATTEN_CHAR_RSP :='606F'H}),
                              md_CMN_CMN_CmAttenCharRsp_001(
                              payloadHeader, acvarfield))) to v_sut_mac;

            if(v_count > par_C_EV_match_retry) {
              alt{
                 [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_ATTEN_CHAR_IND := '606E'H}),
                                          mdw_SECC_CMN_CmAttenCharInd_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(),
                                          par_testSystem_mac, vc_RunID, ?)))
                                          -> sender v_sut_mac {

                  setverdict(fail,"CM_ATTEN_CHAR.IND message was repeated, but " &
                                  "v_count > par_C_EV_match_retry.");
                }
                [] tc_TT_match_response.timeout {
                  setverdict(pass,"TT_match_response timeout. " &
                                  "The total number of retries is reached, " &
                                  "the Matching process shall " &
                                  "be considered as FAILED");
                  pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_CHAR_RSP :='606F'H}),
                                    md_CMN_CMN_CmAttenCharRsp_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    md_CMN_CMN_Acvarfield_001(
                                    par_testSystem_mac, vc_RunID))))
                                    to v_sut_mac;

                  tc_TT_match_response.start(par_TT_match_response);
                  pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_SLAC_MATCH_REQ := '607C'H}),
                                    md_CMN_CMN_CmSlacMatchReq_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    par_testSystem_mac, vc_sut_mac, vc_RunID)))
                                    to vc_sut_mac;

                  alt {
                         []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                                  CM_SLAC_MATCH_CNF := '607D'H}),
                                                 md_CMN_CMN_CmSlacMatchCnf_001(
                                                 m_CMN_CMN_SlacPayloadHeader_001(),
                                                 par_testSystem_mac, vc_sut_mac,
                                                 vc_RunID, ?, ?))) {

                             setverdict(fail,"CM_SLAC_MATCH.CNF message " &
                                             "was not expected. " &
                                             "Repetition limit was reached.");
                             tc_TT_match_response.stop;
                         }
```

```
                            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                                     CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                                // CM_ATTEN_CHAR.IND messages will be ignored!
                                repeat;
                            }
                            [] a_SECC_processPLCLinkNotifications_001();
                            [] pt_SLAC_Port.receive {
                               setverdict(fail, "Invalid message type or content " &
                                                "was received.");
                            }
                            [] tc_TT_match_response.timeout {
                                setverdict(pass,"TT_match_response timeout. " &
                                                 "Matching process is " &
                                                 "considered as FAILED.");
                            }
                            [] tc_TT_matching_repetition.timeout {
                               log("TT_matching_repetition timeout - " &
                                   "No new matching process can be started, " &
                                   "if the current matching process fails.");
                               repeat;
                            }
                       }
                    }
                  }
               }
               else{
                   repeat;
               }
          }
          [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_SLAC_PARM_CNF := '6065'H}),?)) {
             // CM_SLAC_PARM.CNF messages will be ignored!
             repeat;
          }
          [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_PROFILE_IND := '6086'H}),?)) {

             // CM_ATTEN_PROFILE.IND messages will be ignored!
             repeat;
          }
          [] a_SECC_processPLCLinkNotifications_001();
          [] pt_SLAC_Port.receive {
             setverdict(fail, "Invalid message type or content was received.");
          }
          [] tc_TT_EV_atten_results.timeout {
             setverdict(fail,"TT_EV_atten_results timeout and no CM_ATTEN_CHAR.IND " &
                              "received - EVSE_NOT_FOUND.");
          }
          [] tc_TT_match_response.timeout {
             setverdict(fail,"TT_match_response timeout. " &
                              "CM_ATTEN_CHAR.IND message was not repeated.");
          }
          [] tc_TT_matching_repetition.timeout {
             log("TT_matching_repetition timeout - " &
                   "No new matching process can be started, " &
                 "if the current matching process fails.");
             repeat;
          }
        }
        return getverdict;
    }

function f_SECC_CMN_TB_VTB_AttenuationCharacterization_006() runs on SECC_Tester
                                                              return verdicttype {

   var MME v_responseMessage;
   var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
   vc_LowestAverageAttenuation := 0.0;

   tc_TT_match_sequence.start(par_TT_match_sequence);

   // wait until tc_TT_match_sequence timer expires
   tc_TT_match_sequence.timeout;

   tc_TT_EV_atten_results.start(par_TT_EV_atten_results);
```

```
        for (var integer i:=0; i<3; i:=i+1) {
            tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_START_ATTEN_CHAR_IND := '606A'H}),
                                  md_CMN_CMN_CmStartAttenCharInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                                  vc_Time_out, '01'H, par_testSystem_mac, vc_RunID)))
                                  to cc_eth_broadcast;

            tc_TP_EV_batch_msg_interval.timeout;
        };

        var integer v_cnt := par_C_EV_match_MNBC;
        for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
            v_cnt := v_cnt -1;
            tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_MNBC_SOUND_IND := '6076'H}),
                                  md_CMN_CMN_CmMnbcSoundInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(),
                                  int2hex(v_cnt,2), vc_RunID, v_source_rnd)))
                                  to cc_eth_broadcast;

            tc_TP_EV_batch_msg_interval.timeout;

        };

        alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_ATTEN_CHAR_IND := '606E'H}),
                                        mdw_SECC_CMN_CmAttenCharInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        par_testSystem_mac, vc_RunID, ?)))
                                        -> value v_responseMessage {

                setverdict(fail,"CM_ATTEN_CHAR.IND message was not expected. " &
                                 "TT_EV_atten_results timer should have expired.");
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_SLAC_PARM_CNF := '6065'H}),?)) {
               // CM_SLAC_PARM.CNF messages will be ignored!
               repeat;
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_ATTEN_PROFILE_IND := '6086'H}),?)) {

               // CM_ATTEN_PROFILE.IND messages will be ignored!
               repeat;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_EV_atten_results.timeout {
               setverdict(pass,"TT_EV_atten_results timeout, Matching process is " &
                                  "considered as FAILED.");
            }
            [] tc_TT_matching_repetition.timeout {
               log("TT_matching_repetition timeout - No new matching process can be started, " &
                   "if the current matching process fails.");
               repeat;
            }
         }
         return getverdict;
    }

    function f_SECC_CMN_TB_VTB_AttenuationCharacterization_007(in template MME_Payload v_payload)
                                                            runs on SECC_Tester
                                                            return verdicttype {

            var MME v_responseMessage;
            var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
        vc_LowestAverageAttenuation := 0.0;

        tc_TP_match_sequence.timeout;
        tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

        for (var integer i:=0; i<3; i:=i+1) {
            tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
            // send invalid CM_START_ATTEN_CHAR.IND message
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_START_ATTEN_CHAR_IND := '606A'H}), v_payload))
                                to cc_eth_broadcast;

            tc_TP_EV_batch_msg_interval.timeout;
        };

        var integer v_cnt := par_C_EV_match_MNBC;
        for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
            v_cnt := v_cnt -1;
            tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_MNBC_SOUND_IND := '6076'H}),
                             md_CMN_CMN_CmMnbcSoundInd_001(
                             m_CMN_CMN_SlacPayloadHeader_001(),
                             int2hex(v_cnt,2), vc_RunID, v_source_rnd)))
                             to cc_eth_broadcast;

            tc_TP_EV_batch_msg_interval.timeout;

        };

        alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_CHAR_IND := '606E'H}),
                                   mdw_SECC_CMN_CmAttenCharInd_001(
                                   m_CMN_CMN_SlacPayloadHeader_001(),
                                   par_testSystem_mac, vc_RunID, ?)))
                                   -> value v_responseMessage {

                setverdict(fail,"CM_ATTEN_CHAR.IND message was not expected. " &
                                  "Invalid CM_START_ATTEN_CHAR.IND " &
                                  "message was sent before.");

            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_SLAC_PARM_CNF := '6065'H}),?)) {
               // CM_SLAC_PARM.CNF messages will be ignored!
               repeat;
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_PROFILE_IND := '6086'H}),?)) {

               // CM_ATTEN_PROFILE.IND messages will be ignored!
               repeat;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_EV_atten_results.timeout {
               setverdict(pass,"TT_EV_atten_results timeout. " &
                                "No valid CM_START_ATTEN_CHAR.IND " &
                                "message was received before.");
            }
            [] tc_TT_matching_repetition.timeout {
               log("TT_matching_repetition timeout - " &
                     "No new matching process can be started, " &
                   "if the current matching process fails.");
               repeat;
            }
         }
         return getverdict;
    }

function f_SECC_CMN_TB_VTB_AttenuationCharacterization_008(in HAL_61851_Listener
                                                        v_HAL_61851_Listener)
                                                        runs on SECC_Tester
                                                        return verdicttype {

    var MME v_responseMessage;
    var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
    vc_LowestAverageAttenuation := 0.0;

         f_SECC_changeValidStateCondition(invalid);
         f_SECC_changeValidFrequencyRange(0,0);
         f_SECC_changeValidDutyCycleRange(100,100);
         deactivate(vc_Default_IEC_61851_ListenerBehavior);
         f_SECC_setState(A,v_HAL_61851_Listener);


    tc_TP_match_sequence.timeout;
    tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

    for (var integer i:=0; i<3; i:=i+1) {
        tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_START_ATTEN_CHAR_IND := '606A'H}),
                          md_CMN_CMN_CmStartAttenCharInd_001(
                          m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                          vc_Time_out, '01'H, par_testSystem_mac, vc_RunID)))
                          to cc_eth_broadcast;

        tc_TP_EV_batch_msg_interval.timeout;
    };

    var integer v_cnt := par_C_EV_match_MNBC;
    for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
        v_cnt := v_cnt -1;
        tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_MNBC_SOUND_IND := '6076'H}),
                          md_CMN_CMN_CmMnbcSoundInd_001(
                          m_CMN_CMN_SlacPayloadHeader_001(), int2hex(v_cnt,2),
                          vc_RunID, v_source_rnd))) to cc_eth_broadcast;

        tc_TP_EV_batch_msg_interval.timeout;

    };

    alt {
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_ATTEN_CHAR_IND := '606E'H}),
                                mdw_SECC_CMN_CmAttenCharInd_001(
                                m_CMN_CMN_SlacPayloadHeader_001(),
                                par_testSystem_mac, vc_RunID, ?)))
                                -> value v_responseMessage {

            setverdict(fail,"CM_ATTEN_CHAR.IND message was not expected." &
                           "CP State A should be detected before.");
        }
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_SLAC_PARM_CNF := '6065'H}),?)) {
           // CM_SLAC_PARM.CNF messages will be ignored!
           repeat;
        }
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_ATTEN_PROFILE_IND := '6086'H}),?)) {

           // CM_ATTEN_PROFILE.IND messages will be ignored!
           repeat;
        }
        [] a_SECC_processPLCLinkNotifications_001();
        [] pt_SLAC_Port.receive {
           setverdict(fail, "Invalid message type or content was received.");
        }
        [] tc_TT_EV_atten_results.timeout {
           setverdict(pass,"TT_EV_atten_results timer has expired, " &
                          "the Matching process was terminated by the SUT.");
         }
         [] tc_TT_matching_repetition.timeout {
            log("TT_matching_repetition timeout - " &
                  "No new matching process can be started, " &
                "if the current matching process fails.");
            repeat;
         }
      }
      return getverdict;
   }

   function f_SECC_CMN_TB_VTB_AttenuationCharacterization_009(in MACAddress_TYPE
                                                            v_macAddress)
                                                            runs on SLAC_Tester
                                                            return verdicttype {

      var MME v_responseMessage;
      var boolean v_repetition := true;
      var integer v_count1 := 0;
      var integer v_count2 := 0;
      var MACAddress_TYPE v_sut_mac;
      var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
      vc_LowestAverageAttenuation := 0.0;

      vc_macAddresList := m_CMN_CMN_EmptyMacAddresList();
      vc_RunID := f_randomHexStringGen(16);
      tc_TT_matching_repetition.start(par_TT_matching_repetition);
      while(v_repetition){

          tc_TT_match_response.start(par_TT_match_response);
          v_count1 := v_count1 + 1;
          pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_SLAC_PARM_REQ := '6064'H}),
                            md_CMN_CMN_CmSlacParmReq_001(
                            m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                            to cc_eth_broadcast;

           alt {
                  []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_SLAC_PARM_CNF := '6065'H}),
                                         md_CMN_CMN_CmSlacParmCnf_001(
                                         v_macAddress ,m_CMN_CMN_SlacPayloadHeader_001(),
                                         vc_RunID)))
                                         -> value v_responseMessage sender v_sut_mac{

                      setverdict(pass,"CM_SLAC_PARM is correct.");
                      vc_macAddresList.macAddressList[v_count2] := v_sut_mac;
                      v_count2 := v_count2 + 1;
                      vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                       cm_slac_parm_cnf.num_sounds;
                      vc_Time_out := v_responseMessage.mme_payload.payload.
                                     cm_slac_parm_cnf.time_out;
                      repeat;
                  }
                  [] a_SECC_processPLCLinkNotifications_002();
                  [] pt_SLAC_Port.receive {
                     setverdict(fail, "Invalid message type or content was received.");
                     v_repetition := false;
                  }
                  [] tc_TT_match_response.timeout {
                      if(isbound(vc_macAddresList.macAddressList)){
                          tc_TP_match_sequence.start(par_TP_match_sequence);
                          v_repetition := false;
                      }
                      else if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                          log("The Matching process is considered as FAILED.");
                          if(tc_TT_matching_repetition.running){
                              log("TT_matching_repetition is still running. " &
                                  "A new Matching process is started.");
                              v_count1 := 0;
                          }
                          else {
                              setverdict(fail, "TT_matching_repetition has expired. " &
                                               "No new Matching process will be started.");
                              v_repetition := false;
                          }
                      }
                }
            }
        }
        if(getverdict == pass) {
            tc_TP_match_sequence.timeout;
            tc_TT_EV_atten_results.start(par_TT_EV_atten_results);

            for (var integer i:=0; i<3; i:=i+1) {
                tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_START_ATTEN_CHAR_IND := '606A'H}),
                                  md_CMN_CMN_CmStartAttenCharInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                                  vc_Time_out, '01'H, v_macAddress, vc_RunID)))
                                  to cc_eth_broadcast;

                tc_TP_EV_batch_msg_interval.timeout;
            };

            var integer v_cnt := par_C_EV_match_MNBC;
            for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
                v_cnt := v_cnt -1;
                tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_MNBC_SOUND_IND := '6076'H}),
                                  md_CMN_CMN_CmMnbcSoundInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), int2hex(v_cnt,2),
                                  vc_RunID, v_source_rnd))) to cc_eth_broadcast;

                tc_TP_EV_batch_msg_interval.timeout;

            };

            var integer v_cnt_pot_evse := 0;

            alt {
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_ATTEN_CHAR_IND := '606E'H}),
                                        mdw_SECC_CMN_CmAttenCharInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        v_macAddress, vc_RunID, ?)))
                                        -> value v_responseMessage sender v_sut_mac {

                    setverdict(pass,"CM_ATTEN_CHAR.IND is correct.");
                    v_cnt_pot_evse := v_cnt_pot_evse + 1;
                    tc_TP_match_sequence.start(par_TP_match_sequence);

                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_CHAR_RSP :='606F'H}),
                                      md_CMN_CMN_CmAttenCharRsp_001(
                                      m_CMN_CMN_SlacPayloadHeader_001(),
                                      md_CMN_CMN_Acvarfield_001(v_macAddress, vc_RunID))))
                                      to v_sut_mac;

                    tc_TP_match_sequence.stop;
                    f_SECC_CMN_setMac(v_responseMessage, v_sut_mac);
                    if(sizeof(vc_macAddresList.macAddressList) == v_cnt_pot_evse) {
                     log("CM_ATTEN_CHAR.IND messages from all EVSEs are received.");
                    }
                    else{repeat;}
                }
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_SLAC_PARM_CNF := '6065'H}),?)) {
                   // CM_SLAC_PARM.CNF messages will be ignored!
                   repeat;
                }
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
                   // CM_ATTEN_PROFILE.IND messages will be ignored!
                   repeat;
                }
                [] a_SECC_processPLCLinkNotifications_002();
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                }
                [] tc_TT_EV_atten_results.timeout {
                   if(v_cnt_pot_evse == 0){
                        setverdict(fail,"TT_EV_atten_results timeout and no CM_ATTEN_CHAR.IND " &
                                        "received - EVSE_NOT_FOUND.");
                   }
                }
                [] tc_TT_matching_repetition.timeout {
                   log("TT_matching_repetition timeout - " &
                       "No new matching process can be started, " &
                       "if the current matching process fails.");
                   repeat;
                }
             }
         }
         return getverdict;
    }

    function f_SECC_CMN_setMac(MME v_responseMessage, MACAddress_TYPE v_sut_mac_temp) runs on
SLAC_Tester {
        var AttenProfile_TYPE v_attenuation_list := v_responseMessage.mme_payload.
                                                payload.cm_atten_char_ind.attenuation_list;
        var float averageAttenuation := f_SECC_CMN_calculateAttenuation(v_attenuation_list);

        log("SUT MAC address: ", v_sut_mac_temp, "Average attenuation: ", averageAttenuation);
        if((averageAttenuation < vc_LowestAverageAttenuation) or
           (vc_LowestAverageAttenuation == 0.0)){
            vc_LowestAverageAttenuation := averageAttenuation;
            vc_sut_mac := v_sut_mac_temp;
            log("An SECC with a lower attenuation could be detected.");
        }
    }

    function f_SECC_CMN_calculateAttenuation(AttenProfile_TYPE v_attenuation_list) runs on
SLAC_Tester
                                                                        return float {
        var integer v_attenuationAdded := 0;

        for (var integer i:=0; i<sizeof(v_attenuation_list); i:=i + 1) {
            v_attenuationAdded := v_attenuationAdded + hex2int(v_attenuation_list.attenuation[i]);
        }

        var float v_averageAttenuation := (int2float(v_attenuationAdded)/
                                          int2float(sizeof(v_attenuation_list)));
        return v_averageAttenuation;
    }

    function f_SECC_CMN_Reset_001(in HAL_61851_Listener v_HAL_61851_Listener) runs on SECC_Tester {

      // initiate restart of the matching process
      all timer.stop;
      v_HAL_61851_Listener.stop;
      v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(false));
      f_SECC_changeValidStateCondition(A);
      f_SECC_changeValidFrequencyRange(0,0);
      f_SECC_changeValidDutyCycleRange(100,100);
      f_SECC_setState(A,v_HAL_61851_Listener);
      f_SECC_setProximity(0);

      f_SECC_CMN_setTXPower(10);

      v_HAL_61851_Listener.stop;
      v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(true));

      if(PICS_CMN_CMN_ChargingMode == aC){
          f_SECC_changeValidFrequencyRange(0,0);
          f_SECC_changeValidDutyCycleRange(100,100);
          vc_validDutyCycleLowerBound2 := 100;
          vc_validDutyCycleUpperBound2 := 100;
      }

      if(PICS_CMN_CMN_PlugType == type1) {
          f_SECC_setProximity(cc_proximity_type1);
      } else {
          if(PICS_CMN_CMN_ChargingMode == aC){
              var integer v_proximity_type2_AC;
              if(PICS_CMN_AC_CableCapability == capability13A) {
                v_proximity_type2_AC := cc_proximity_type2_AC_13A;
            } else if(PICS_CMN_AC_CableCapability == capability20A) {
                v_proximity_type2_AC := cc_proximity_type2_AC_20A;
            } else if(PICS_CMN_AC_CableCapability == capability32A) {
                v_proximity_type2_AC := cc_proximity_type2_AC_32A;
            } else {
                v_proximity_type2_AC := cc_proximity_type2_AC_63A;
            }
            f_SECC_setProximity(v_proximity_type2_AC);
        } else {
            f_SECC_setProximity(cc_proximity_type2_DC);
        }
      }
      sleep((par_CMN_waitForNextHAL));
      f_SECC_changeValidStateCondition(valid_Matching);
      f_SECC_setState(vc_state,v_HAL_61851_Listener);
      tc_TT_matching_repetition.start(par_TT_matching_repetition);
    }

    function f_SECC_CMN_compareAttenuationValues_001(in AttenProfile_TYPE
                                                     v_attenuation_list1,
                                                     in AttenProfile_TYPE
                                                     v_attenuation_list2) {
        var integer v_meanAttenuation1;
        var integer v_meanAttenuation2;

        v_meanAttenuation1 := f_SECC_CMN_calculateMeanOfAttenuationValues_001(
                              v_attenuation_list1);
        v_meanAttenuation2 := f_SECC_CMN_calculateMeanOfAttenuationValues_001(
                              v_attenuation_list2);

        if((v_meanAttenuation2 - v_meanAttenuation1) < par_SECC_attenuationDeviation) {
            setverdict(fail, "Invalid attenuation values were detected. The deviation " &
                             "was smaller than 'par_SECC_attenuationDeviation'.");
        } else {
            setverdict(pass, "Valid attenuation values were detected. The deviation " &
                             "was greater or equal than 'par_SECC_attenuationDeviation'.");
        }
    }

    function f_SECC_CMN_calculateMeanOfAttenuationValues_001(in AttenProfile_TYPE
                                                             v_attenuation_list)
                                                             return integer{

        var integer v_result := 0;

        for (var integer i:=0; i<sizeof(v_attenuation_list); i:=i + 1) {
            v_result := v_result + hex2int(v_attenuation_list.attenuation[i]);
        }
        v_result := v_result/sizeof(v_attenuation_list);
        return v_result;
    }
}
```

E.1.3 SECC functions for CmValidate

```
module TestBehavior_SECC_CmValidate {

    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Templates_CMN_CmValidate all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_CmSlacParm all;
    import from Templates_SECC_CmAttenCharInd all;
    import from Templates_CMN_CmStartAttenCharInd all;
    import from Templates_CMN_CmMnbcSoundInd all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from TestBehavior_SECC_AttenuationCharacterization all;
    import from LibFunctions_15118_3 all;
    import from Services_HAL_61851 all;
    import from Templates_CMN_CmAttenCharRsp all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Services_PLCLinkStatus all;
    import from Pics_15118 all;
    import from Timer_15118 all;
```

```
// SECC Tester

function f_SECC_CMN_TB_VTB_CmValidate_001(in HAL_61851_Listener v_HAL_61851_Listener,
                                          in verdicttype v_vct)
                                          runs on SECC_Tester return verdicttype {

    var boolean v_repetition := true;
    var integer v_count1 := 0;
    var MME v_responseMessage;

    if(f_checkValidToggleConfig()) {

        while(v_repetition){

            tc_TT_match_response.start(par_TT_match_response);
            v_count1 := v_count1 + 1;
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_VALIDATE_REQ := '6078'H}),
                              m_CMN_CMN_CmValidateReq_001()))
                              to vc_sut_mac;

             alt {
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_VALIDATE_CNF := '6079'H}),
                                            md_CMN_CMN_CmValidateCnf_001(?)))
                                            -> value v_responseMessage {

                        tc_TT_match_response.stop;
                        var hexstring v_result := v_responseMessage.mme_payload.
                                            payload.cm_validate_cnf.vcVarField.result;
                        v_repetition := f_checkResultFieldStep1(v_result,
                                                                v_repetition,
                                                                v_vct);
                    }
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                        tc_TT_match_response.stop;
                        tc_TT_match_response.start(par_TT_match_response);
                        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_ATTEN_CHAR_RSP :='606F'H}),
                                          md_CMN_CMN_CmAttenCharRsp_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(),
                                          md_CMN_CMN_Acvarfield_001(
                                          par_testSystem_mac, vc_RunID))))
                                          to vc_sut_mac;

                        log("A further CM_ATTEN_CHAR.IND message was received. " &
                            "A new CM_ATTEN_CHAR.RSP has to be send.");
                        repeat;
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(v_vct, "Invalid message type or content was received.");
                       v_repetition := false;
                    }
                    [] tc_TT_match_response.timeout {
                        log("TT_match_response timeout.");
                        if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                            setverdict(v_vct,"The repetition limit is reached. " &
                                             "The Matching process is considered as FAILED.");
                            v_repetition := false;
                        } else {
                          log("The repetition limit is not reached, " &
                              "a new CM_VALIDATE.REQ message will be send.");
                        }
                    }
                    [] tc_TT_matching_repetition.timeout {
                        log("TT_matching_repetition timeout - " &
                            "No new matching process can be started, " &
                            "if the current matching process fails.");
                       repeat;
                    }
             }
        }
```

```
if(getverdict() == pass){
    // encoding vald toggle time
    var hexstring v_pilotTimer := int2hex(float2int((
                            par_TP_EV_vald_toggle * 10.0) - 1.0),2);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                  md_CMN_CMN_SlacMmeCmnHeader_001({
                  CM_VALIDATE_REQ := '6078'H}),
                  md_CMN_CMN_CmValidateReq_002(v_pilotTimer)))
                  to vc_sut_mac;

    tc_TP_EV_vald_toggle.start(par_TP_EV_vald_toggle + par_CMN_Transmission_Delay);

    // start BCB toggle sequence
    for (var integer i:=0; i<C_vald_nb_toggles; i:=i + 1) {
        // B toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_setState(B,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // C toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_changeValidStateCondition(C);
        f_SECC_setState(C,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // B toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_changeValidStateCondition(B);
        f_SECC_setState(B,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
    }

    alt {
           [] tc_TP_EV_vald_toggle.timeout {
              tc_TT_match_response.start(par_TT_match_response);
              alt {
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_CNF := '6079'H}),
                                      md_CMN_CMN_CmValidateCnf_002(?,?)))
                                      -> value v_responseMessage {

                       tc_TT_match_response.stop;
                       var hexstring v_result := v_responseMessage.mme_payload.
                                         payload.cm_validate_cnf.vcVarField.
                                         result;
                       var ToggleNum_TYPE v_toggle_num := v_responseMessage.mme_payload.
                                              payload.cm_validate_cnf.
                                              vcVarField.toggle_num;
                       f_checkResultFieldStep2(v_result, v_vct);
                       if(getverdict() == pass){
                         if(hex2int(v_toggle_num) == C_vald_nb_toggles) {
                            setverdict(pass,"EVSE_FOUND, the number of detected " &
                                       "BCB toggles is correct.");
                         }
                         else {
                            setverdict(v_vct,"The number of detected BCB " &
                                        "toggles is not correct.");
                         }
                       }
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(v_vct, "Invalid message type or content " &
                                         "was received.");
                    }
                    [] tc_TT_match_response.timeout {
                       setverdict(v_vct,"TT_match_response timeout. " &
                                        "The Validation process will be stopped.");
                    }
                    [] tc_TT_matching_repetition.timeout {
                        log("TT_matching_repetition timeout - " &
                            "No new matching process can be started, " &
                            "if the current matching process fails.");
                       repeat;
                    }
              }
        }
    }
}
```

```
        }
        else {
            setverdict(inconc,"Invalid BCB toggle configuration. Check module parameter.");
        }
        return getverdict;
      }

    function f_SECC_CMN_TB_VTB_CmValidate_002() runs on SECC_Tester return verdicttype {

        var boolean v_repetition := true;
        var integer v_count1 := 0;
        var MME v_responseMessage;

        if(f_checkValidToggleConfig()) {

            while(v_repetition){

                tc_TT_match_response.start(par_TT_match_response);
                v_count1 := v_count1 + 1;
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_REQ := '6078'H}),
                                  m_CMN_CMN_CmValidateReq_001()))
                                  to vc_sut_mac;

                 alt {
                        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                                CM_VALIDATE_CNF := '6079'H}),
                                                md_CMN_CMN_CmValidateCnf_001(?)))
                                                -> value v_responseMessage {

                            tc_TT_match_response.stop;
                            var hexstring v_result := v_responseMessage.mme_payload.payload.
                                                      cm_validate_cnf.vcVarField.result;
                            v_repetition := f_checkResultFieldStep1(v_result, v_repetition,
                                                                    fail);
                        }
                        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                                CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                            tc_TT_match_response.stop;
                            tc_TT_match_response.start(par_TT_match_response);
                            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                              md_CMN_CMN_SlacMmeCmnHeader_001({
                                              CM_ATTEN_CHAR_RSP :='606F'H}),
                                              md_CMN_CMN_CmAttenCharRsp_001(
                                              m_CMN_CMN_SlacPayloadHeader_001(),
                                              md_CMN_CMN_Acvarfield_001(
                                              par_testSystem_mac, vc_RunID))))
                                              to vc_sut_mac;

                            log("A further CM_ATTEN_CHAR.IND message was received. " &
                                "A new CM_ATTEN_CHAR.RSP has to be send.");
                            repeat;
                        }
                        [] a_SECC_processPLCLinkNotifications_001();
                        [] pt_SLAC_Port.receive {
                           setverdict(fail, "Invalid message type or content was received.");
                           v_repetition := false;
                        }
                        [] tc_TT_match_response.timeout {
                            log("TT_match_response timeout.");
                            if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                                setverdict(fail,"The repetition limit is reached. " &
                                                "The Matching process is considered as FAILED.");
                                v_repetition := false;
                            } else {
                              log("The repetition limit is not reached, " &
                                  "a new CM_VALIDATE.REQ message will be send.");
                            }
                        }
                        [] tc_TT_matching_repetition.timeout {
                            log("TT_matching_repetition timeout - " &
                                "No new matching process can be started, " &
                                "if the current matching process fails.");
                           repeat;
                        }
                }
            }
            if(getverdict() == pass){
                var hexstring v_pilotTimer := int2hex(float2int(par_TP_EV_vald_toggle * 10.0),2);
                tc_TT_match_response.start(par_TT_match_response);
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_REQ := '6078'H}),
                                  md_CMN_CMN_CmValidateReq_002('00'H)))
                                  to vc_sut_mac;

                alt {
                       [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                                  CM_VALIDATE_CNF := '6079'H}),
                                               md_CMN_CMN_CmValidateCnf_001(?))) {

                          setverdict(pass,"Step 1 CM_VALIDATE.CNF message was repeated.");
                       }
                       [] a_SECC_processPLCLinkNotifications_001();
                       [] pt_SLAC_Port.receive {
                          setverdict(fail, "Invalid message type or content was received.");
                       }
                       [] tc_TT_match_response.timeout {
                            setverdict(fail,"TT_match_response timeout. " &
                                            "Step 1 CM_VALIDATE.CNF message was not repeated.");
                       }
                       [] tc_TT_matching_repetition.timeout {
                           log("TT_matching_repetition timeout - " &
                                  "No new matching process can be started, " &
                               "if the current matching process fails.");
                            repeat;
                       }
                }
            }
        }
        else {
            setverdict(inconc,"Invalid BCB toggle configuration. Check module parameter.");
        }
        return getverdict;
      }

    function f_SECC_CMN_TB_VTB_CmValidate_003(in HAL_61851_Listener v_HAL_61851_Listener)
                                            runs on SECC_Tester return verdicttype {

        var MME v_responseMessage;
        var MACAddress_TYPE v_address;
        var integer v_count := 0;

        tc_TT_match_response.start(par_TT_match_response);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_VALIDATE_REQ := '6078'H}),
                          m_CMN_CMN_CmValidateReq_001()))
                          to vc_sut_mac;

        alt{
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_CNF := '6079'H}),
                                     md_CMN_CMN_CmValidateCnf_001(?)))
                                     -> value v_responseMessage {

                tc_TT_match_response.stop;
                if(v_count > 0){
                    setverdict(pass,"CM_VALIDATE.CNF message was repeated.",v_count);
                }
                v_count := v_count + 1;
                tc_TT_match_sequence.start(par_TT_match_sequence + par_CMN_Transmission_Delay);

                if(v_count > par_C_EV_match_retry) {
                  alt{
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                             md_CMN_CMN_SlacMmeCmnHeader_001({
                                             CM_VALIDATE_CNF := '6079'H}),
                                             md_CMN_CMN_CmValidateCnf_001(?)))
                                             -> value v_responseMessage {
```

```
            setverdict(fail,"CM_VALIDATE.CNF message was repeated, but v_count > " &
                              "par_C_EV_match_retry.");
          }
          [] tc_TT_match_sequence.timeout {
            setverdict(pass,"TT_match_sequence timeout. " &
                             "The total number of retries is reached, the Validation " &
                             "process shall be considered as FAILED");

            // encoding vald toggle time
            var hexstring v_pilotTimer := int2hex(float2int((
                                            par_TP_EV_vald_toggle * 10.0) - 1.0),2);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                               md_CMN_CMN_SlacMmeCmnHeader_001({
                               CM_VALIDATE_REQ := '6078'H}),
                             md_CMN_CMN_CmValidateReq_002(v_pilotTimer)))
                             to vc_sut_mac;

            tc_TP_EV_vald_toggle.start(par_TP_EV_vald_toggle + par_CMN_Transmission_Delay);

            // start BCB toggle sequence
            for (var integer i:=0; i<C_vald_nb_toggles; i:=i + 1) {
                // B toggle
                tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
                f_SECC_setState(B,v_HAL_61851_Listener);
                tc_TP_EV_vald_state_duration.timeout;
                // C toggle
                tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
                f_SECC_changeValidStateCondition(C);
                f_SECC_setState(C,v_HAL_61851_Listener);
                tc_TP_EV_vald_state_duration.timeout;
                // B toggle
                tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
                f_SECC_changeValidStateCondition(B);
                f_SECC_setState(B,v_HAL_61851_Listener);
                tc_TP_EV_vald_state_duration.timeout;
            }

            alt {
                   [] tc_TP_EV_vald_toggle.timeout {
                      tc_TT_match_response.start(par_TT_match_response);
                      alt {
                            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                                       CM_VALIDATE_CNF := '6079'H}),
                                                     md_CMN_CMN_CmValidateCnf_002(?,?))) {

                               tc_TT_match_response.stop;
                               setverdict(fail,"CM_VALIDATE.CNF message " &
                                                "was not expected, the Matching" &
                                                " process shall be considered " &
                                                "as FAILED.");
                            }
                            [] a_SECC_processPLCLinkNotifications_001();
                            [] pt_SLAC_Port.receive {
                               setverdict(fail, "Invalid message type or content " &
                                                 "was received.");
                            }
                            [] tc_TT_match_response.timeout {
                               setverdict(pass,"TT_match_response timeout. " &
                                                "Matching process is considered " &
                                                "as FAILED.");
                            }
                            [] tc_TT_matching_repetition.timeout {
                               log("TT_matching_repetition timeout - " &
                                    "No new matching process can be started, " &
                                   "if the current matching process fails.");
                               repeat;
                            }
                      }
                   }
            }
          }
        }
      }
      else{
          repeat;
      }
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_ATTEN_CHAR_IND := '606E'H}),?)) {

            tc_TT_match_response.stop;
            tc_TT_match_response.start(par_TT_match_response);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_ATTEN_CHAR_RSP :='606F'H}),
                              md_CMN_CMN_CmAttenCharRsp_001(
                              m_CMN_CMN_SlacPayloadHeader_001(),
                              md_CMN_CMN_Acvarfield_001(
                              par_testSystem_mac, vc_RunID))))
                              to vc_sut_mac;

            log("A further CM_ATTEN_CHAR.IND message was received. " &
                "A new CM_ATTEN_CHAR.RSP has to be send.");
            repeat;
         }
         [] a_SECC_processPLCLinkNotifications_001();
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
         }
         [] tc_TT_match_response.timeout {
            setverdict(fail,"TT_match_response timer timeout. " &
                            "No CM_VALIDATE.CNF message was received.");
         }
         [] tc_TT_match_sequence.timeout {
            setverdict(fail,"TT_match_sequence timeout. " &
                            "CM_VALIDATE.CNF message was not repeated.");
         }
         [] tc_TT_matching_repetition.timeout {
            log("TT_matching_repetition timeout - No new matching process can be started, " &
                "if the current matching process fails.");
            repeat;
         }
     }
     return getverdict;
}

function f_SECC_CMN_TB_VTB_CmValidate_004(in HAL_61851_Listener v_HAL_61851_Listener,
                                          in template(present) MME_Payload mmePayload)
                                          runs on SECC_Tester return verdicttype {

    var MME v_responseMessage;
    var MACAddress_TYPE v_address;
    var integer v_count := 0;

    tc_TT_match_response.start(par_TT_match_response);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_VALIDATE_REQ := '6078'H}),
                      m_CMN_CMN_CmValidateReq_001()))
                      to vc_sut_mac;

    alt{
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_VALIDATE_CNF := '6079'H}),
                                md_CMN_CMN_CmValidateCnf_001(?)))
                                -> value v_responseMessage {

            tc_TT_match_response.stop;
            if(v_count > 0){
                setverdict(pass,"CM_VALIDATE.CNF message was repeated.",v_count);
            }
            v_count := v_count + 1;
            tc_TT_match_sequence.start(par_TT_match_sequence + par_CMN_Transmission_Delay);
            // send invalid step 2 CM_VALIDATE.REQ message
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_VALIDATE_REQ := '6078'H}),
                              mmePayload)) to vc_sut_mac;

            if(v_count > par_C_EV_match_retry) {
              alt{
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_VALIDATE_CNF := '6079'H}),
                              md_CMN_CMN_CmValidateCnf_001(?)))
                              -> value v_responseMessage {

          setverdict(fail,"CM_VALIDATE.CNF message was repeated, but v_count > " &
                          "par_C_EV_match_retry.");
        }
        [] tc_TT_match_sequence.timeout {
          setverdict(pass,"TT_match_sequence timeout. " &
                          "The total number of retries is reached, the Validation " &
                          "process shall be considered as FAILED.");

          // encoding vald toggle time
          var hexstring v_pilotTimer := int2hex(float2int((
                                    par_TP_EV_vald_toggle * 10.0) - 1.0),2);

          pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_VALIDATE_REQ := '6078'H}),
                          md_CMN_CMN_CmValidateReq_002(v_pilotTimer)))
                          to vc_sut_mac;

          tc_TP_EV_vald_toggle.start(par_TP_EV_vald_toggle + par_CMN_Transmission_Delay);

          // start BCB toggle sequence
          for (var integer i:=0; i<C_vald_nb_toggles; i:=i + 1) {
              // B toggle
              tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
              f_SECC_setState(B,v_HAL_61851_Listener);
              tc_TP_EV_vald_state_duration.timeout;
              // C toggle
              tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
              f_SECC_changeValidStateCondition(C);
              f_SECC_setState(C,v_HAL_61851_Listener);
              tc_TP_EV_vald_state_duration.timeout;
              // B toggle
              tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
              f_SECC_changeValidStateCondition(B);
              f_SECC_setState(B,v_HAL_61851_Listener);
              tc_TP_EV_vald_state_duration.timeout;
          }

          alt {
             [] tc_TP_EV_vald_toggle.timeout {
                tc_TT_match_response.start(par_TT_match_response);
                alt {
                      [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_VALIDATE_CNF := '6079'H}),
                                          md_CMN_CMN_CmValidateCnf_002(?,?))) {

                         tc_TT_match_response.stop;
                         setverdict(fail,"CM_VALIDATE.CNF message " &
                                         "was not expected, the Matching " &
                                         "process shall be considered as FAILED.");
                      }
                      [] a_SECC_processPLCLinkNotifications_001();
                      [] pt_SLAC_Port.receive {
                         setverdict(fail, "Invalid message type or content " &
                                          "was received.");
                      }
                      [] tc_TT_match_response.timeout {
                         setverdict(pass,"TT_match_response timeout. " &
                                         "Matching process was " &
                                         "considered as FAILED.");
                      }
                      [] tc_TT_matching_repetition.timeout {
                         log("TT_matching_repetition timeout - " &
                              "No new matching process can be started, " &
                             "if the current matching process fails.");
                         repeat;
                      }
                }
             }
           }
        }
      }
    }
    else{
        repeat;
```

```
            }
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_ATTEN_CHAR_IND := '606E'H}),?)) {

            tc_TT_match_response.stop;
            tc_TT_match_response.start(par_TT_match_response);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_ATTEN_CHAR_RSP :='606F'H}),
                              md_CMN_CMN_CmAttenCharRsp_001(
                              m_CMN_CMN_SlacPayloadHeader_001(),
                              md_CMN_CMN_Acvarfield_001(
                              par_testSystem_mac, vc_RunID))))
                              to vc_sut_mac;

            log("A further CM_ATTEN_CHAR.IND message was received. " &
                "A new CM_ATTEN_CHAR.RSP has to be send.");
            repeat;
         }
         [] a_SECC_processPLCLinkNotifications_001();
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
         }
         [] tc_TT_match_response.timeout {
            setverdict(fail,"TT_match_response timeout. " &
                            "CM_VALIDATE.CNF message was not repeated.");
         }
         [] tc_TT_matching_repetition.timeout {
            log("TT_matching_repetition timeout - " &
                "No new matching process can be started, " &
                "if the current matching process fails.");
            repeat;
         }
      }
      return getverdict;
  }

  function f_SECC_CMN_TB_VTB_CmValidate_005(in template(present) MME_Payload mmePayload)
                                           runs on SECC_Tester return verdicttype {

      var MME v_responseMessage;
      var MACAddress_TYPE v_address;

      tc_TT_match_response.start(par_TT_match_response);
      pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                        md_CMN_CMN_SlacMmeCmnHeader_001({
                        CM_VALIDATE_REQ := '6078'H}),
                        m_CMN_CMN_CmValidateReq_001()))
                        to vc_sut_mac;

      alt{
          [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(?)))
                                  -> value v_responseMessage {

              tc_TT_match_response.stop;
              setverdict(pass,"CM_VALIDATE.CNF message was successful.");

              tc_TT_match_sequence.start(par_TT_match_sequence + par_CMN_Transmission_Delay);
              // send step 2 CM_VALIDATE.REQ message with a result field unequal than 'ready'
              pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_VALIDATE_REQ := '6078'H}),
                                mmePayload)) to vc_sut_mac;

              alt{
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_VALIDATE_CNF := '6079'H}),
                                          md_CMN_CMN_CmValidateCnf_001(?)))
                                          -> value v_responseMessage {

                     setverdict(fail,"CM_VALIDATE.CNF message was repeated, " &
                                      "but Validation process " &
                                 "shall be considered as FAILED.");
                 }
                 [] a_SECC_processPLCLinkNotifications_001();
                 [] pt_SLAC_Port.receive {
                    setverdict(fail, "Invalid message type or content was received.");
                 }
                 [] tc_TT_match_sequence.timeout {
                    setverdict(pass,"TT_match_sequence timeout. " &
                                 "The Validation process is considered as FAILED.");
                 }
                 [] tc_TT_matching_repetition.timeout {
                     log("TT_matching_repetition timeout - " &
                         "No new matching process can be started, " &
                         "if the current matching process fails.");
                    repeat;
                 }
             }
          }
          [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_ATTEN_CHAR_IND := '606E'H}),?)) {

             tc_TT_match_response.stop;
             tc_TT_match_response.start(par_TT_match_response);
             pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_ATTEN_CHAR_RSP :='606F'H}),
                             md_CMN_CMN_CmAttenCharRsp_001(
                             m_CMN_CMN_SlacPayloadHeader_001(),
                             md_CMN_CMN_Acvarfield_001(
                             par_testSystem_mac, vc_RunID))))
                             to vc_sut_mac;

             log("A further CM_ATTEN_CHAR.IND message was received. " &
                 "A new CM_ATTEN_CHAR.RSP has to be send.");
             repeat;
          }
          [] a_SECC_processPLCLinkNotifications_001();
          [] pt_SLAC_Port.receive {
             setverdict(fail, "Invalid message type or content was received.");
          }
          [] tc_TT_match_response.timeout {
             setverdict(fail,"TT_match_response timeout. " &
                             "CM_VALIDATE.CNF message was not repeated.");
          }
          [] tc_TT_matching_repetition.timeout {
             log("TT_matching_repetition timeout - " &
                 "No new matching process can be started, " &
                 "if the current matching process fails.");
             repeat;
          }
      }
      return getverdict;
}

function f_SECC_CMN_TB_VTB_CmValidate_006() runs on SECC_Tester return verdicttype {

    var boolean v_repetition := true;
    var integer v_count1 := 0;
    var MME v_responseMessage;

    if(f_checkValidToggleConfig()) {

        if(tc_TP_EVSE_match_session.running) {
           tc_TP_EVSE_match_session.stop;

           while(v_repetition){

                tc_TT_match_response.start(par_TT_match_response);
                v_count1 := v_count1 + 1;
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_VALIDATE_REQ := '6078'H}),
                                 m_CMN_CMN_CmValidateReq_001()))
                                 to vc_sut_mac;

                 alt {
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
```

```
                                        CM_VALIDATE_CNF := '6079'H}),
                                        md_CMN_CMN_CmValidateCnf_001(?)))
                                        -> value v_responseMessage {

                            tc_TT_match_response.stop;
                            var hexstring v_result := v_responseMessage.mme_payload.
                                                      payload.cm_validate_cnf.
                                                      vcVarField.result;
                            v_repetition := f_checkResultFieldStep1(v_result, v_repetition,
                                                                    fail);
                        }
                        [] a_SECC_processPLCLinkNotifications_001();
                        [] pt_SLAC_Port.receive {
                           setverdict(fail, "Invalid message type or content was received.");
                           v_repetition := false;
                        }
                        [] tc_TT_match_response.timeout {
                            log("TT_match_response timeout.");
                            if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                                setverdict(fail,"The repetition limit is reached. " &
                                                "The Matching process is considered as FAILED.");
                                v_repetition := false;
                            } else {
                              log("The repetition limit is not reached, " &
                                  "a new CM_VALIDATE.REQ message will be send.");
                            }
                        }
                        [] tc_TT_matching_repetition.timeout {
                            log("TT_matching_repetition timeout - " &
                                "No new matching process can be started, " &
                                "if the current matching process fails.");
                           repeat;
                        }
                     }
                }
            }
            else {
                setverdict(inconc, "The tc_TP_EVSE_match_session timer has expired. " &
                                   "The validation process can not be continued.");
            }
        }
        else {
            setverdict(inconc,"Invalid BCB toggle configuration. Check module parameter.");
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_CmValidate_007(in HAL_61851_Listener v_HAL_61851_Listener)
                                          runs on SECC_Tester return verdicttype {

        var MME v_responseMessage;

        // encoding vald toggle time
        var hexstring v_pilotTimer := int2hex(float2int((
                                      par_TP_EV_vald_toggle * 10.0) - 1.0),2);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_VALIDATE_REQ := '6078'H}),
                          md_CMN_CMN_CmValidateReq_002(v_pilotTimer)))
                          to vc_sut_mac;

        tc_TP_EV_vald_toggle.start(par_TP_EV_vald_toggle + par_CMN_Transmission_Delay);

        // start BCB toggle sequence
        for (var integer i:=0; i<C_vald_nb_toggles; i:=i + 1) {
            // B toggle
            tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
            f_SECC_setState(B,v_HAL_61851_Listener);
            tc_TP_EV_vald_state_duration.timeout;
            // C toggle
            tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
            f_SECC_changeValidStateCondition(C);
            f_SECC_setState(C,v_HAL_61851_Listener);
            tc_TP_EV_vald_state_duration.timeout;
            // B toggle
            tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
            f_SECC_changeValidStateCondition(B);
            f_SECC_setState(B,v_HAL_61851_Listener);
            tc_TP_EV_vald_state_duration.timeout;
        }

        alt {
           [] tc_TP_EV_vald_toggle.timeout {
              tc_TT_match_response.start(par_TT_match_response);
              alt {
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_VALIDATE_CNF := '6079'H}),
                                            md_CMN_CMN_CmValidateCnf_002(?,?)))
                                            -> value v_responseMessage {

                       tc_TT_match_response.stop;
                       var hexstring v_result := v_responseMessage.mme_payload.
                                                 payload.cm_validate_cnf.vcVarField.
                                                 result;
                       var ToggleNum_TYPE v_toggle_num := v_responseMessage.mme_payload.
                                                          payload.cm_validate_cnf.
                                                          vcVarField.toggle_num;
                       f_checkResultFieldStep2(v_result, fail);
                       if(getverdict() == pass){
                         if(hex2int(v_toggle_num) == C_vald_nb_toggles) {
                            setverdict(pass,"EVSE_FOUND, the number of detected " &
                                            "BCB toggles is correct.");
                         }
                         else {
                            setverdict(fail,"The number of detected BCB " &
                                            "toggles is not correct.");
                         }
                       }
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(fail, "Invalid message type or content was received.");
                    }
                    [] tc_TT_match_response.timeout {
                       setverdict(fail,"TT_match_response timeout. " &
                                     "Validation process will be stopped.");
                    }
                    [] tc_TT_matching_repetition.timeout {
                        log("TT_matching_repetition timeout - " &
                            "No new matching process can be started, " &
                            "if the current matching process fails.");
                       repeat;
                    }
                }
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_CmValidate_008(in Result_TYPE p_result)
                                          runs on SECC_Tester
                                          return verdicttype {

        var boolean v_repetition := true;
        var integer v_count1 := 0;
        var MME v_responseMessage;

        while(v_repetition){

            tc_TT_match_response.start(par_TT_match_response);
            v_count1 := v_count1 + 1;
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_VALIDATE_REQ := '6078'H}),
                              m_CMN_CMN_CmValidateReq_001()))
                              to vc_sut_mac;

             alt {
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_VALIDATE_CNF := '6079'H}),
                                            md_CMN_CMN_CmValidateCnf_001(
                                            p_result))) {
                        tc_TT_match_response.stop;
                    }
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                    tc_TT_match_response.stop;
                    tc_TT_match_response.start(par_TT_match_response);
                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_ATTEN_CHAR_RSP :='606F'H}),
                                    md_CMN_CMN_CmAttenCharRsp_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    md_CMN_CMN_Acvarfield_001(
                                    par_testSystem_mac, vc_RunID))))
                                    to vc_sut_mac;

                    log("A further CM_ATTEN_CHAR.IND message was received. " &
                        "A new CM_ATTEN_CHAR.RSP has to be send.");
                    repeat;
                }
                [] a_SECC_processPLCLinkNotifications_001();
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                   v_repetition := false;
                }
                [] tc_TT_match_response.timeout {
                    log("TT_match_response timeout.");
                    if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                        setverdict(fail,"The repetition limit is reached. " &
                                        "The Matching process is considered as FAILED.");
                        v_repetition := false;
                    } else {
                      log("The repetition limit is not reached, " &
                          "a new CM_VALIDATE.REQ message will be send.");
                    }
                }
                [] tc_TT_matching_repetition.timeout {
                    log("TT_matching_repetition timeout - " &
                        "No new matching process can be started, " &
                        "if the current matching process fails.");
                   repeat;
                }
         }
    }
    return getverdict;
  }

// SLAC Tester

function f_SECC_CMN_TB_VTB_CmValidate_009() runs on SLAC_Tester return verdicttype {

    var boolean v_repetition := true;
    var integer v_count1 := 0;
    var MME v_responseMessage;

    while(v_repetition){

        tc_TT_match_response.start(par_TT_match_response);
        v_count1 := v_count1 + 1;
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_VALIDATE_REQ := '6078'H}),
                          m_CMN_CMN_CmValidateReq_001()))
                          to vc_sut_mac;

         alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_VALIDATE_CNF := '6079'H}),
                                    md_CMN_CMN_CmValidateCnf_001(
                                    par_cmValidate_result_notReady))) {
                setverdict(pass,"CM_VALIDATE.CNF message with 'notReady' is correct. " &
                                "SUT has indicated that it is temporary busy.");
                tc_TT_match_response.stop;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
               v_repetition := false;
            }
            [] tc_TT_match_response.timeout {
                log("TT_match_response timeout.");
                if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                    setverdict(fail,"The repetition limit is reached. " &
                                    "The Matching process is considered as FAILED.");
                    v_repetition := false;
                } else {
                  log("The repetition limit is not reached, " &
                      "a new CM_VALIDATE.REQ message will be send.");
                }
            }
            [] tc_TT_matching_repetition.timeout {
                log("TT_matching_repetition timeout - " &
                    "No new matching process can be started, " &
                    "if the current matching process fails.");
               repeat;
            }
        }
    }
    return getverdict;
}

function f_SECC_CMN_TB_VTB_CmValidate_010(in HAL_61851_Listener v_HAL_61851_Listener)
                                          runs on SECC_Tester return verdicttype {

         f_SECC_changeValidStateCondition(invalid);
         f_SECC_changeValidFrequencyRange(0,0);
         f_SECC_changeValidDutyCycleRange(100,100);
         deactivate(vc_Default_IEC_61851_ListenerBehavior);
         f_SECC_setState(A,v_HAL_61851_Listener);


    tc_TT_match_response.start(par_TT_match_response);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_VALIDATE_REQ := '6078'H}),
                      m_CMN_CMN_CmValidateReq_001()))
                      to vc_sut_mac;

     alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_VALIDATE_CNF := '6079'H}),
                                    md_CMN_CMN_CmValidateCnf_001(?))) {
                setverdict(fail,"CM_VALIDATE.CNF message was not expected." &
                                "CP State A should be detected before.");

            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                tc_TT_match_response.stop;
                tc_TT_match_response.start(par_TT_match_response);
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_RSP :='606F'H}),
                                  md_CMN_CMN_CmAttenCharRsp_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(),
                                  md_CMN_CMN_Acvarfield_001(
                                  par_testSystem_mac, vc_RunID))))
                                  to vc_sut_mac;

                log("A further CM_ATTEN_CHAR.IND message was received. " &
                    "A new CM_ATTEN_CHAR.RSP has to be send.");
                repeat;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_match_response.timeout {
                setverdict(pass,"TT_match_response timer has expired, " &
                                "the Matching process was terminated " &
                                "by the SUT.");
            }
            [] tc_TT_matching_repetition.timeout {
                log("TT_matching_repetition timeout - " &
                    "No new matching process can be started, " &
                    "if the current matching process fails.");
               repeat;
            }
        }
        return getverdict;
     }

   function f_SECC_CMN_TB_VTB_CmValidatePreCondition_001() runs on SLAC_Tester
                                                           return verdicttype {

       var MME v_responseMessage;
       var boolean v_repetition := true;
       var MACAddress_TYPE v_sut_mac;
       var integer v_count1 := 0;
       var integer v_count2 := 0;
       var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
       vc_LowestAverageAttenuation := 0.0;

       vc_macAddresList := m_CMN_CMN_EmptyMacAddresList();
       vc_RunID := f_randomHexStringGen(16);

       while(v_repetition){

           tc_TT_match_response.start(par_TT_match_response);
           v_count1 := v_count1 + 1;
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(md_CMN_CMN_SlacMmeCmnHeader_001({
                                                   CM_SLAC_PARM_REQ := '6064'H}),
                                                   md_CMN_CMN_CmSlacParmReq_001(
                                                   m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                                   to cc_eth_broadcast;

            alt {
               []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_SLAC_PARM_CNF := '6065'H}),
                                      md_CMN_CMN_CmSlacParmCnf_001(par_testSystem_mac,
                                      m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                      -> value v_responseMessage sender v_sut_mac {

                   setverdict(pass,"CM_SLAC_PARM is correct.");
                   vc_macAddresList.macAddressList[v_count2] := v_sut_mac;
                   v_count2 := v_count2 + 1;
                   vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                    cm_slac_parm_cnf.num_sounds;
                   vc_Time_out := v_responseMessage.mme_payload.payload.
                                  cm_slac_parm_cnf.time_out;
                   repeat;
               }
               [] a_SECC_processPLCLinkNotifications_002();
               [] pt_SLAC_Port.receive {
                  setverdict(fail, "Invalid message type or content was received.");
                  v_repetition := false;
               }
               [] tc_TT_match_response.timeout {
                   if(sizeof(vc_macAddresList.macAddressList) > 0){
                       tc_TP_match_sequence.start(par_TP_match_sequence);
                       v_repetition := false;
                   }
                   else if(v_count1 mod (par_C_EV_match_retry+1) == 0){
                       log("The Matching process is considered as FAILED.");
                       if(tc_TT_matching_repetition.running){
                           log("TT_matching_repetition is still running. " &
                               "A new Matching process is started.");
                           v_count1 := 0;
                       }
                       else {
                           setverdict(fail, "TT_matching_repetition has expired. " &
                                            "No new Matching process will be started.");
                           v_repetition := false;
                       }
                   }
               }
            }
       }

       if(getverdict == pass) {
           tc_TP_match_sequence.timeout;
           tc_TT_EV_atten_results.start(par_TT_EV_atten_results);
```

```
for (var integer i:=0; i<3; i:=i+1) {
    tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_START_ATTEN_CHAR_IND := '606A'H}),
                      md_CMN_CMN_CmStartAttenCharInd_001(
                      m_CMN_CMN_SlacPayloadHeader_001(), vc_Num_sounds,
                      vc_Time_out, '01'H, par_testSystem_mac, vc_RunID)))
                      to cc_eth_broadcast;

    tc_TP_EV_batch_msg_interval.timeout;
};

var integer v_cnt := par_C_EV_match_MNBC;
for (var integer i:=0; i<par_C_EV_match_MNBC; i:=i+1) {
    v_cnt := v_cnt -1;
    tc_TP_EV_batch_msg_interval.start(par_TP_EV_batch_msg_interval);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_MNBC_SOUND_IND := '6076'H}),
                      md_CMN_CMN_CmMnbcSoundInd_001(
                      m_CMN_CMN_SlacPayloadHeader_001(), int2hex(v_cnt,2),
                      vc_RunID, v_source_rnd))) to cc_eth_broadcast;

    tc_TP_EV_batch_msg_interval.timeout;

};

var integer v_cnt_pot_evse := 0;

 alt {
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_ATTEN_CHAR_IND := '606E'H}),
                                mdw_SECC_CMN_CmAttenCharInd_001(
                                m_CMN_CMN_SlacPayloadHeader_001(),
                                par_testSystem_mac, vc_RunID, ?)))
                                -> value v_responseMessage sender v_sut_mac {

            setverdict(pass,"CM_ATTEN_CHAR.IND is correct.");
            v_cnt_pot_evse := v_cnt_pot_evse + 1;
            tc_TP_match_sequence.start(par_TP_match_sequence);

            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_ATTEN_CHAR_RSP :='606F'H}),
                              md_CMN_CMN_CmAttenCharRsp_001(
                              m_CMN_CMN_SlacPayloadHeader_001(),
                              md_CMN_CMN_Acvarfield_001(par_testSystem_mac, vc_RunID))))
                              to v_sut_mac;

            tc_TP_match_sequence.stop;
            f_SECC_CMN_setMac(v_responseMessage, v_sut_mac);
            if(sizeof(vc_macAddresList.macAddressList) == v_cnt_pot_evse) {
             log("CM_ATTEN_CHAR.IND messages from all EVSEs are received.");
            }
            else{repeat;}
        }
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_SLAC_PARM_CNF := '6065'H}),?)) {
           // CM_SLAC_PARM.CNF messages will be ignored!
           repeat;
        }
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
           // CM_ATTEN_PROFILE.IND messages will be ignored!
           repeat;
        }
        [] a_SECC_processPLCLinkNotifications_002();
        [] pt_SLAC_Port.receive {
           setverdict(fail, "Invalid message type or content was received.");
        }
        [] tc_TT_EV_atten_results.timeout {
           if(v_cnt_pot_evse == 0){
                setverdict(fail,"TT_EV_atten_results timeout and no CM_ATTEN_CHAR.IND " &
                                "received - EVSE_NOT_FOUND.");
```

```
                    }
                 }
                 [] tc_TT_matching_repetition.timeout {
                    log("TT_matching_repetition timeout - " &
                        "No new matching process can be started, " &
                        "if the current matching process fails.");
                    repeat;
                 }
              }
           }
           return getverdict;
        }

        function f_checkResultFieldStep1(hexstring v_result,
                                         boolean v_repetition,
                                         in verdicttype v_vct)
                                         runs on SECC_Tester
                                         return boolean {

          if (v_result == par_cmValidate_result_notReady) {
              sleep(0.5);
          }
          else if (v_result == par_cmValidate_result_ready) {
              setverdict(pass,"CM_VALIDATE.CNF is correct. The EV will " &
                              "send the step 2 CM_VALIDATE.REQ " &
                              "with a Timer value which covers the whole " &
                              "BCB toggle sequence.");
              v_repetition := false;
          }
          else if((v_result == par_cmValidate_result_success) or
                  (v_result == par_cmValidate_result_failure)) {
              setverdict(v_vct,"Invalid result code, the EV will stop " &
                               "the Validation process with " &
                               "the current EVSE.");
              v_repetition := false;
          }
          else if (v_result == par_cmValidate_result_notRequired) {
              setverdict(inconc,"The validation process is not required.");
              v_repetition := false;
          }
          else {
              setverdict(v_vct,"Unkwnown result format.");
              v_repetition := false;
          }
          return v_repetition;
        }

      function f_checkResultFieldStep2(hexstring v_result,
                                         in verdicttype v_vct)
                                         runs on SECC_Tester {


        if (v_result == par_cmValidate_result_success) {
            setverdict(pass,"CM_VALIDATE.CNF is correct. The EV will compare " &
                            "the ToggleNum field of the CM_VALIDATE.CNF message " &
                            "with the number of BCB toggles executed.");
        }
        else if((v_result == par_cmValidate_result_notReady) or
                (v_result == par_cmValidate_result_ready) or
                (v_result == par_cmValidate_result_failure) or
                (v_result == par_cmValidate_result_notRequired)) {

            setverdict(v_vct,"Invalid result code, the EV will stop " &
                             "the Validation process with " &
                             "the current EVSE.");
        }
        else {
            setverdict(v_vct,"Unkwnown result format.");
        }
      }

      function f_checkValidToggleConfig() return boolean {
          if((int2float(C_vald_nb_toggles) * par_TP_EV_vald_state_duration * 3.0) <
    par_TP_EV_vald_toggle) {
              return true;
          }
          return false;
      }
```

```
}
```

E.1.4 SECC functions for CmSlacMatch

```
module TestBehavior_SECC_CmSlacMatch {

    import from Timer_15118_3 all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from Templates_CMN_CmSlacMatch all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from DataStructure_HAL_61851 all;
    import from Services_HAL_61851 all;
    import from Templates_CMN_CmAttenCharRsp all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Services_PLCLinkStatus all;
    import from Timer_15118 all;

    function f_SECC_CMN_TB_VTB_CmSlacMatch_001(in verdicttype v_vct)
                                              runs on SECC_Tester
                                              return verdicttype {

        var MME v_responseMessage;
        var boolean v_repetition := true;
        var integer v_counter := 0;

        while(v_repetition){

            tc_TT_match_response.start(par_TT_match_response);
            v_counter := v_counter + 1;
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_SLAC_MATCH_REQ := '607C'H}),
                              md_CMN_CMN_CmSlacMatchReq_001(
                              m_CMN_CMN_SlacPayloadHeader_001(),
                              par_testSystem_mac, vc_sut_mac, vc_RunID)))
                              to vc_sut_mac;

             alt {
                    []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_MATCH_CNF := '607D'H}),
                                     md_CMN_CMN_CmSlacMatchCnf_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(),
                                     par_testSystem_mac, vc_sut_mac, vc_RunID, ?, ?)))
                                     -> value v_responseMessage {

                        setverdict(pass,"CM_SLAC_MATCH is correct.");
                        vc_Nid := v_responseMessage.mme_payload.payload.cm_slac_match_cnf.nid;
                        vc_Nmk := v_responseMessage.mme_payload.payload.cm_slac_match_cnf.nmk;
                        v_repetition := false;
                        tc_TT_match_response.stop;
                    }
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                        tc_TT_match_response.stop;
                        tc_TT_match_response.start(par_TT_match_response);
                        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_ATTEN_CHAR_RSP :='606F'H}),
                                    md_CMN_CMN_CmAttenCharRsp_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    md_CMN_CMN_Acvarfield_001(
                                    par_testSystem_mac, vc_RunID))))
                                    to vc_sut_mac;

                        log("A further CM_ATTEN_CHAR.IND message was received. " &
                            "A new CM_ATTEN_CHAR.RSP has to be send.");
                        repeat;
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
```

```
                    setverdict(v_vct, "Invalid message type or content was received.");
                    v_repetition := false;
                 }
                 [] tc_TT_match_response.timeout {

                     log("TT_match_response timeout.");
                     if(v_counter mod (par_C_EV_match_retry+1) == 0){
                         setverdict(v_vct,"The repetition limit is reached. " &
                                          "The Matching process is considered " &
                                          "as FAILED.");
                         v_repetition := false;
                     } else {
                       log("The repetition limit is not reached, " &
                           "a new CM_SLAC_MATCH.REQ message will be send.");
                     }
                 }
                 [] tc_TT_matching_repetition.timeout {
                     log("TT_matching_repetition timeout - " &
                         "No new matching process can be started, " &
                         "if the current matching process fails.");
                    repeat;
                 }
          }
      }
       return getverdict;
    }

  function f_SECC_CMN_TB_VTB_CmSlacMatch_002() runs on SECC_Tester return verdicttype {

      tc_TT_match_response.start(par_TT_match_response);
      pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                        md_CMN_CMN_SlacMmeCmnHeader_001({
                        CM_SLAC_MATCH_REQ := '607C'H}),
                        md_CMN_CMN_CmSlacMatchReq_001(
                        m_CMN_CMN_SlacPayloadHeader_001(),
                        par_testSystem_mac, vc_sut_mac, vc_RunID)))
                        to vc_sut_mac;

       alt {
              []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_MATCH_CNF := '607D'H}),
                                     md_CMN_CMN_CmSlacMatchCnf_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(),
                                     par_testSystem_mac, vc_sut_mac, vc_RunID, ?, ?))) {
                  repeat;
              }
              [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                  tc_TT_match_response.stop;
                  tc_TT_match_response.start(par_TT_match_response);
                  pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_ATTEN_CHAR_RSP :='606F'H}),
                                    md_CMN_CMN_CmAttenCharRsp_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    md_CMN_CMN_Acvarfield_001(
                                    par_testSystem_mac, vc_RunID))))
                                    to vc_sut_mac;

                  log("A further CM_ATTEN_CHAR.IND message was received. " &
                      "A new CM_ATTEN_CHAR.RSP has to be send.");
                  repeat;
              }
              [] a_SECC_processPLCLinkNotifications_001();
              [] pt_SLAC_Port.receive {
                 setverdict(fail, "Invalid message type or content was received.");
              }
              [] tc_TT_match_response.timeout {
                  // retransmitting CM_SLAC_MATCH.REQ message
                  tc_TT_match_response.start(par_TT_match_response);
                  pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_SLAC_MATCH_REQ := '607C'H}),
                                    md_CMN_CMN_CmSlacMatchReq_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                  par_testSystem_mac, vc_sut_mac, vc_RunID)))
                                  to vc_sut_mac;

                  alt {
                      []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                             md_CMN_CMN_SlacMmeCmnHeader_001({
                                             CM_SLAC_MATCH_CNF := '607D'H}),
                                             md_CMN_CMN_CmSlacMatchCnf_001(
                                             m_CMN_CMN_SlacPayloadHeader_001(), par_testSystem_mac,
                                             vc_sut_mac, vc_RunID, ?, ?))) {

                          tc_TT_match_response.stop;
                          setverdict(pass,"TT_match_response timeout. " &
                                          "CM_SLAC_MATCH.CNF was retransmitted.");
                      }
                      [] a_SECC_processPLCLinkNotifications_001();
                      [] pt_SLAC_Port.receive {
                         setverdict(fail, "Invalid message type or content was received.");
                      }
                      [] tc_TT_match_response.timeout {
                          setverdict(fail,"TT_match_response timeout. " &
                                          "CM_SLAC_MATCH.CNF was not retransmitted.");
                      }
                  }
              }
              [] tc_TT_matching_repetition.timeout {
                  log("TT_matching_repetition timeout - " &
                      "No new matching process can be started, " &
                      "if the current matching process fails.");
                 repeat;
              }
       }
       return getverdict;
     }

   function f_SECC_CMN_TB_VTB_CmSlacMatch_003() runs on SECC_Tester return verdicttype {

       var charstring v_timer_name;

       if(PIXIT_SECC_CMN_CmValidate == cmValidate) {
           v_timer_name := "TT_match_sequence";
           tc_TT_match_sequence.start(par_TT_match_sequence +
                                      par_CMN_Transmission_Delay);
           // wait until TT_match_sequence timer expires
           alt {
               [] tc_TT_match_sequence.timeout;
               [] a_SECC_processPLCLinkNotifications_001();
               [] pt_SLAC_Port.receive {
                  setverdict(fail, "Invalid message type or content was received.");
               }
               [] tc_TT_matching_repetition.timeout {
                   log("TT_matching_repetition timeout - " &
                       "No new matching process can be started, " &
                       "if the current matching process fails.");
                  repeat;
               }
           }

       }
       else {
           v_timer_name := "TT_EVSE_match_session";
           tc_TT_EVSE_match_session.start(par_TT_EVSE_match_session +
                                          par_CMN_Transmission_Delay);
           // wait until TT_EVSE_match_session timer expires
           alt {
               [] tc_TT_EVSE_match_session.timeout;
               [] a_SECC_processPLCLinkNotifications_001();
               [] pt_SLAC_Port.receive {
                  setverdict(fail, "Invalid message type or content was received.");
               }
               [] tc_TT_matching_repetition.timeout {
                   log("TT_matching_repetition timeout - " &
                       "No new matching process can be started, " &
                       "if the current matching process fails.");
                  repeat;
               }
           }
       }
```

```
    tc_TT_match_response.start(par_TT_match_response);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_SLAC_MATCH_REQ := '607C'H}),
                      md_CMN_CMN_CmSlacMatchReq_001(
                      m_CMN_CMN_SlacPayloadHeader_001(),
                      par_testSystem_mac, vc_sut_mac, vc_RunID)))
                      to vc_sut_mac;

     alt {
            []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_SLAC_MATCH_CNF := '607D'H}),
                                   md_CMN_CMN_CmSlacMatchCnf_001(
                                   m_CMN_CMN_SlacPayloadHeader_001(),
                                   par_testSystem_mac, vc_sut_mac, vc_RunID, ?, ?))) {

                setverdict(fail,"CM_SLAC_MATCH.CNF was not expected, " & v_timer_name &
                                " timer has been expired.");
                tc_TT_match_response.stop;
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                tc_TT_match_response.stop;
                tc_TT_match_response.start(par_TT_match_response);
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_RSP :='606F'H}),
                                  md_CMN_CMN_CmAttenCharRsp_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(),
                                  md_CMN_CMN_Acvarfield_001(
                                  par_testSystem_mac, vc_RunID))))
                                  to vc_sut_mac;

                log("A further CM_ATTEN_CHAR.IND message was received. " &
                    "A new CM_ATTEN_CHAR.RSP has to be send.");
                repeat;
            }
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_match_response.timeout {
                setverdict(pass,"TT_match_response timeout. " &
                                "Matching process is considered as FAILED.");
            }
            [] tc_TT_matching_repetition.timeout {
                log("TT_matching_repetition timeout - " &
                    "No new matching process can be started, " &
                    "if the current matching process fails.");
               repeat;
            }
     }
     return getverdict;
  }

function f_SECC_CMN_TB_VTB_CmSlacMatch_004(in template MME_Payload v_payload)
                                          runs on SECC_Tester return verdicttype {

    timer v_timer;
    var charstring v_timer_name;

    if(PIXIT_SECC_CMN_CmValidate == cmValidate) {
        v_timer_name := "TT_match_sequence";
        v_timer.start(par_TT_match_sequence + par_CMN_Transmission_Delay);
    }
    else {
        v_timer_name := "TT_EVSE_match_session";
        v_timer.start(par_TT_EVSE_match_session + par_CMN_Transmission_Delay);
    }

    // send invalid CM_SLAC_MATCH.REQ message
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_SLAC_MATCH_REQ := '607C'H}), v_payload))
                      to vc_sut_mac;
        alt {
           []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_MATCH_CNF := '607D'H}),?)) {

               setverdict(fail,"Invalid CM_SLAC_MATCH.REQ messages shall be ignored.");
               tc_TT_EVSE_match_session.stop;
           }
           [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_ATTEN_CHAR_IND := '606E'H}),?)) {

               tc_TT_match_response.stop;
               tc_TT_match_response.start(par_TT_match_response);
               pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_RSP :='606F'H}),
                               md_CMN_CMN_CmAttenCharRsp_001(
                               m_CMN_CMN_SlacPayloadHeader_001(),
                               md_CMN_CMN_Acvarfield_001(
                               par_testSystem_mac, vc_RunID))))
                               to vc_sut_mac;

               log("A further CM_ATTEN_CHAR.IND message was received. " &
                      "A new CM_ATTEN_CHAR.RSP has to be send.");
               repeat;
           }
           [] a_SECC_processPLCLinkNotifications_001();
           [] pt_SLAC_Port.receive {
              setverdict(fail, "Invalid message type or content was received.");
           }
           [] v_timer.timeout {

            tc_TT_match_response.start(par_TT_match_response);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_SLAC_MATCH_REQ := '607C'H}),
                            md_CMN_CMN_CmSlacMatchReq_001(
                            m_CMN_CMN_SlacPayloadHeader_001(),
                            par_testSystem_mac, vc_sut_mac, vc_RunID)))
                            to vc_sut_mac;

             alt {
                    []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_SLAC_MATCH_CNF := '607D'H}),
                                          md_CMN_CMN_CmSlacMatchCnf_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(),
                                          par_testSystem_mac, vc_sut_mac, vc_RunID, ?, ?))) {

                        setverdict(fail,"CM_SLAC_MATCH.CNF was not expected, "
                                        & v_timer_name & " has been expired.");
                        tc_TT_match_response.stop;
                    }
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                        // CM_ATTEN_CHAR.IND messages will be ignored!
                        repeat;
                    }
                    [] a_SECC_processPLCLinkNotifications_001();
                    [] pt_SLAC_Port.receive {
                       setverdict(fail, "Invalid message type or content was received.");
                    }
                    [] tc_TT_match_response.timeout {
                        setverdict(pass,"TT_match_response timeout. " &
                                        "Matching process is considered as FAILED.");
                    }
                    [] tc_TT_matching_repetition.timeout {
                        log("TT_matching_repetition timeout - " &
                            "No new matching process can be started, " &
                            "if the current matching process fails.");
                       repeat;
                    }
             }
           }
         }
         return getverdict;
      }

    function f_SECC_CMN_TB_VTB_CmSlacMatch_005(in HAL_61851_Listener v_HAL_61851_Listener)
                                          runs on SECC_Tester return verdicttype {

            f_SECC_changeValidStateCondition(invalid);
            f_SECC_changeValidFrequencyRange(0,0);
            f_SECC_changeValidDutyCycleRange(100,100);
            deactivate(vc_Default_IEC_61851_ListenerBehavior);
            f_SECC_setState(A,v_HAL_61851_Listener);


        tc_TT_match_response.start(par_TT_match_response);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_SLAC_MATCH_REQ := '607C'H}),
                          md_CMN_CMN_CmSlacMatchReq_001(
                          m_CMN_CMN_SlacPayloadHeader_001(),
                          par_testSystem_mac, vc_sut_mac, vc_RunID)))
                          to vc_sut_mac;

         alt {
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SLAC_MATCH_CNF := '607D'H}),
                                       md_CMN_CMN_CmSlacMatchCnf_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(),
                                       par_testSystem_mac, vc_sut_mac,
                                       vc_RunID, ?, ?))) {

                    setverdict(fail,"CM_SLAC_MATCH.CNF message was not expected." &
                                    "CP State A should be detected before.");
                    tc_TT_match_response.stop;
                }
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_ATTEN_CHAR_IND := '606E'H}),?)) {

                    tc_TT_match_response.stop;
                    tc_TT_match_response.start(par_TT_match_response);
                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_CHAR_RSP :='606F'H}),
                                      md_CMN_CMN_CmAttenCharRsp_001(
                                      m_CMN_CMN_SlacPayloadHeader_001(),
                                      md_CMN_CMN_Acvarfield_001(
                                      par_testSystem_mac, vc_RunID))))
                                      to vc_sut_mac;

                    log("A further CM_ATTEN_CHAR.IND message was received. " &
                        "A new CM_ATTEN_CHAR.RSP has to be send.");
                    repeat;
                }
                [] a_SECC_processPLCLinkNotifications_001();
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                }
                [] tc_TT_match_response.timeout {
                    setverdict(pass,"TT_match_response timer has expired, " &
                                    "the Matching process was terminated " &
                                    "by the SUT.");
                }
                [] tc_TT_matching_repetition.timeout {
                    log("TT_matching_repetition timeout - " &
                        "No new matching process can be started, " &
                        "if the current matching process fails.");
                   repeat;
                }
         }
         return getverdict;
      }
}
```

E.1.5 SECC functions for CmSetKey

```
module TestBehavior_SECC_CmSetKey {
    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Templates_CMN_CmSetKey all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from Timer_15118 all;
    import from Services_PLCLinkStatus all;

    function f_SECC_CMN_TB_VTB_CmSetKey_001(in boolean useTimer) runs on SECC_Tester
                                                                  return verdicttype {

        timer t1 := par_CMN_setKey;

        if(useTimer) {
            t1.start;
        }

        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_SET_KEY_REQ := '6008'H}),
                          md_CMN_CMN_CmSetKeyReq_001(vc_Nid, vc_Nmk)))
                          to par_testSystem_plc_node_mac;

         alt {
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_SET_KEY_CNF := '6009'H}),
                                        mdw_CMN_CMN_CmSetKeyCnf_001('01'H))) {

                   setverdict(pass,"CM_SET_KEY is correct.");
                }
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_SLAC_MATCH_CNF := '607D'H}),?)) {
                   repeat;
                }
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SET_KEY_CNF := '6009'H}),
                                       mdw_CMN_CMN_CmSetKeyCnf_001('00'H))) {

                    setverdict(inconc,"CM_SET_KEY is incorrect. " &
                                      "The PLC node could not set the key.");
                }
                [] a_SECC_processPLCLinkNotifications_001();
                [] pt_SLAC_Port.receive {
                   setverdict(inconc, "Invalid message type or content was received.");
                }
                [] tc_TT_match_join.timeout {
                   setverdict(inconc,"CM_SET_KEY timeout.");
                }
                [useTimer == true] t1.timeout {
                   setverdict(inconc,"CM_SET_KEY timeout.");
                }
         }
         return getverdict;
      }
}
```

E.1.6 SECC functions for PLCLinkStatus

```
module TestBehavior_SECC_PLCLinkStatus {

    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from Pics_15118 all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmSlacMatch all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from Services_PLCLinkStatus all;
    import from Services_HAL_61851 all;
    import from Templates_CMN_CmSlacParm all;
    import from TestBehavior_SECC_CmSlacParm all;
import from TestBehavior_SECC_CmSetKey all;
import from LibFunctions_15118_3 all;
import from Timer_15118 all;
import from Pixit_15118 all;
import from DataStructure_SDP all;
import from Pixit_15118_2 all;
import from TestBehavior_SECC_SDP all;
import from Templates_CMN_CmNwStats all;

function f_SECC_CMN_TB_VTB_PLCLinkStatus_001(in verdicttype v_vct)
                                             runs on SECC_Tester
                                             return verdicttype {

    var verdicttype v_verdict;

    v_verdict := f_SECC_getPLCLinkEstablishment(v_vct);

    if (v_verdict == pass) {
        setverdict(pass, "The data link was established by the SUT.");
    }
    else {
        setverdict(v_vct, "The data link could not be established " &
                          "by the SUT.");
    }
    return getverdict;
}

function f_SECC_CMN_TB_VTB_PLCLinkStatus_002(in HAL_61851_Listener
                                             v_HAL_61851_Listener)
                                             runs on SECC_Tester
                                             return verdicttype {

    var verdicttype v_verdict;

    // set state A
        f_SECC_changeValidStateCondition(invalid);
        f_SECC_changeValidFrequencyRange(0,0);
        f_SECC_changeValidDutyCycleRange(100,100);
        deactivate(vc_Default_IEC_61851_ListenerBehavior);
        f_SECC_setState(A,v_HAL_61851_Listener);
    v_verdict := f_SECC_getPLCLinkTermination(par_TP_match_leave +
                                              par_CMN_Transmission_Delay,
                                              fail);

    if (v_verdict == pass) {
        setverdict(pass, "The data link was terminated by the SUT.");
    }
    else {
        setverdict(fail, "The data link did not terminated by the SUT.");
    }
    return getverdict;
}

function f_SECC_CMN_TB_VTB_PLCLinkStatus_003() runs on SECC_Tester
                                               return verdicttype  {

    vc_RunID := f_randomHexStringGen(16);

    tc_TT_match_response.start(par_TT_match_response);
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_SLAC_PARM_REQ := '6064'H}),
                      md_CMN_CMN_CmSlacParmReq_001(
                      m_CMN_CMN_SlacPayloadHeader_001(),
                      vc_RunID))) to cc_eth_broadcast;

    alt {
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_SLAC_PARM_CNF := '6065'H}),
                                         md_CMN_CMN_CmSlacParmCnf_001(
                                         par_testSystem_mac,
                                         m_CMN_CMN_SlacPayloadHeader_001(),
                                         vc_RunID))) {

            setverdict(fail,"CM_SLAC_PARM message was not expected, " &
                            "no SLAC messages should " &
                            "be send in state 'Matched'.");
            }
            [] pt_SLAC_Port.receive {
                setverdict(fail,"Invalid message type or content was received.");
            }
            [] tc_TT_match_response.timeout {
                setverdict(pass,"TT_match_response timeout. SUT did " &
                                "not respond to a CM_SLAC_PARM.REQ message, " &
                                "if it is in state 'Matched'.");
            }
            [] tc_TT_matching_repetition.timeout {
                log("TT_matching_repetition timeout - " &
                    "No new matching process can be started, " &
                    "if the current matching process fails.");
                repeat;
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_PLCLinkStatus_004(in HAL_61851_Listener
                                                 v_HAL_61851_Listener)
                                                 runs on SECC_Tester
                                                 return verdicttype  {

        var verdicttype v_verdict := pass;

        v_HAL_61851_Listener.stop;
        v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(true));

        if(PICS_CMN_CMN_ChargingMode == aC){
            f_SECC_changeValidDutyCycleRange(10,96);
            vc_validDutyCycleLowerBound2 := 10;
            vc_validDutyCycleUpperBound2 := 96;
        }

        // B toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_setState(B,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // C toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_changeValidStateCondition(C);
        f_SECC_setState(C,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // B toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_changeValidStateCondition(B);
        f_SECC_setState(B,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;

        f_SECC_getPLCLinkEstablishmentAfterSleepMode(pass, fail, fail);

        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_PLCLinkStatus_005(in HAL_61851_Listener
                                                 v_HAL_61851_Listener)
                                                 runs on SECC_Tester
                                                 return verdicttype  {

        var verdicttype v_verdict := pass;

        v_HAL_61851_Listener.stop;
        v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(true));

        f_SECC_setIsConfirmationFlagDC();
        if(PICS_CMN_CMN_ChargingMode == aC){
            f_SECC_changeValidDutyCycleRange(10,96);
            vc_validDutyCycleLowerBound2 := 10;
            vc_validDutyCycleUpperBound2 := 96;
        }

        v_verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                            (PICS_CMN_CMN_WakeUp -
                                             PIXIT_CMN_CMN_WakeUp + 5.0),
                                             fail);

        if (getverdict != pass) {
            log("The SUT did not initiate a wake-up " &
                "within 'PICS_CMN_CMN_WakeUp'.");
        }

        f_SECC_getPLCLinkEstablishmentAfterSleepMode(pass, fail, fail);

        if (getverdict == pass) {
            var Security_TYPE v_security := cc_hexTls;
            if(PICS_CMN_CMN_IdentificationMode == eIM and
               PIXIT_SECC_CMN_TLS == false) {
                v_security := cc_hexTcp;
            }
            v_verdict := f_SECC_CMN_TB_VTB_SDP_001(v_security, fail);
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_PLCLinkStatus_006(in HAL_61851_Listener
                                                 v_HAL_61851_Listener)
                                                 runs on SECC_Tester
                                                 return verdicttype {

        var verdicttype v_verdict := pass;

        v_HAL_61851_Listener.stop;
        v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(true));

        if (v_verdict == pass) {

            // B toggle
            tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
            f_SECC_setState(B,v_HAL_61851_Listener);
            tc_TP_EV_vald_state_duration.timeout;
            // C toggle
            tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
            f_SECC_changeValidStateCondition(C);
            f_SECC_setState(C,v_HAL_61851_Listener);
            tc_TP_EV_vald_state_duration.timeout;
            // B toggle
            tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
            f_SECC_changeValidStateCondition(B);
            f_SECC_setState(B,v_HAL_61851_Listener);
            tc_TP_EV_vald_state_duration.timeout;

            tc_T_conn_max_comm.start(par_T_conn_max_comm);

            f_SECC_changeValidStateCondition(EorF);
            f_SECC_changeValidFrequencyRange(0,0);
            f_SECC_changeValidDutyCycleRange(0,0);
            f_SECC_setIsConfirmationFlagVoltage();

            f_SECC_getPLCLinkEstablishmentAfterSleepMode(fail, pass, fail);

            timer statetimer := (par_T_conn_max_comm - tc_T_conn_max_comm.read) +
                                 par_CMN_HAL_Timeout;

            f_SECC_confirmState(EorF, v_HAL_61851_Listener, statetimer);

            if (getverdict != pass) {
                setverdict(fail, "The SUT did not apply CP State E or F.");
            }
            else {
                tc_T_step_EF.start(par_T_step_EF_min - cc_offset);
                alt {
                    [] tc_T_step_EF.timeout {}
                    [] pt_SLAC_Port.receive {
                        setverdict
                        (fail, "Invalid message type or content was received.");
                    }
                }

                f_SECC_changeValidStateCondition(B);
                f_SECC_setIsConfirmationFlagDC();
                f_SECC_changeValidFrequencyRange(980,1020);
                if (PICS_CMN_CMN_ChargingMode == aC) {
                    f_SECC_changeValidDutyCycleRange(10,96);
                    vc_validDutyCycleLowerBound1 := 3;
                    vc_validDutyCycleUpperBound1 := 7;
                    vc_validDutyCycleLowerBound2 := 10;
                vc_validDutyCycleUpperBound2 := 96;
            }

            tc_T_step_EF.start(par_T_step_EF_max -
                              (par_T_step_EF_min - cc_offset));
            alt {
                [] tc_T_step_EF.timeout;
                [] a_SECC_DCDetection(pt_HAL_61851_Internal_Port,
                                      vc_validDutyCycleLowerBound1,
                                      vc_validDutyCycleUpperBound1,
                                      vc_validDutyCycleLowerBound2,
                                      vc_validDutyCycleUpperBound2) {
                   vc_confirmDC := true;
                }
                [] pt_SLAC_Port.receive {
                    setverdict
                    (fail, "Invalid message type or content was received.");
                }
            }
            if(not vc_confirmDC) {
                v_verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                                     par_T_conn_max_comm,
                                                     fail);
            }
        }
    }

    if (getverdict == pass) {
        f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
    }
    return getverdict;
}

function f_SECC_CMN_TB_VTB_PLCLinkStatus_007(in HAL_61851_Listener
                                             v_HAL_61851_Listener)
                                             runs on SECC_Tester
                                             return verdicttype {

    var verdicttype v_verdict := pass;

    sleep(par_CMN_waitForConnectionLoss);

    f_SECC_setIsConfirmationFlagDC();
    f_SECC_changeValidFrequencyRange(0,0);
    f_SECC_changeValidDutyCycleRange(100,100);

    // generate new Nid and Nmk
    vc_Nmk := f_randomHexStringGen(32);
    vc_Nid := fx_generateNID(vc_Nmk);
    v_verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(true);
    if (v_verdict == pass) {
        v_verdict := f_SECC_getPLCLinkTermination(par_TP_match_leave,
                                                  fail);
    }

    if (v_verdict == pass) {
        v_verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                             par_T_conn_max_comm,
                                             fail);

        f_SECC_changeValidStateCondition(EorF);
        f_SECC_changeValidDutyCycleRange(0,0);
        f_SECC_setIsConfirmationFlagVoltage();
    }

    if (v_verdict == pass) {
        timer statetimer := par_CMN_HAL_Timeout;
        v_verdict := f_SECC_confirmState(EorF, v_HAL_61851_Listener,
                                         statetimer);
    }

    if (getverdict != pass) {
        setverdict(fail, "The SUT did not apply CP State E or F.");
    }
    else {
        tc_T_step_EF.start(par_T_step_EF_min - cc_offset);
        alt {
            [] tc_T_step_EF.timeout {}
            [] a_SECC_processPLCLinkNotifications_001();
```

```
                [] pt_SLAC_Port.receive {
                    setverdict
                    (fail, "Invalid message type or content was received.");
                }
            }

            f_SECC_changeValidStateCondition(B);
            f_SECC_setIsConfirmationFlagDC();
            vc_validDutyCycleLowerBound1 := 3;
            vc_validDutyCycleUpperBound1 := 7;
            if (PICS_CMN_CMN_ChargingMode == aC) {
                f_SECC_changeValidDutyCycleRange(10,100);
                vc_validDutyCycleLowerBound2 := 10;
                vc_validDutyCycleUpperBound2 := 100;
            } else {
                f_SECC_changeValidDutyCycleRange(100,100);
                vc_validDutyCycleLowerBound2 := 100;
                vc_validDutyCycleUpperBound2 := 100;
            }

            tc_T_step_EF.start(par_T_step_EF_max -
                              (par_T_step_EF_min - cc_offset));
            alt {
                [] tc_T_step_EF.timeout;
                [] a_SECC_DCDetection(pt_HAL_61851_Internal_Port,
                                      vc_validDutyCycleLowerBound1,
                                      vc_validDutyCycleUpperBound1,
                                      vc_validDutyCycleLowerBound2,
                                      vc_validDutyCycleUpperBound2) {
                   vc_confirmDC := true;
                }
                [] pt_SLAC_Port.receive {
                    setverdict
                    (fail, "Invalid message type or content was received.");
                }
            }
            if(not vc_confirmDC) {
                v_verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                                     par_T_conn_max_comm,
                                                     fail);
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_PLCLinkStatus_008() runs on SECC_Tester
                                                   return verdicttype {

        tc_TP_match_leave.start(par_TP_match_leave);
        alt {
            [] tc_TP_match_leave.timeout {}
            [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or " &
                                 "content was received.");
            }
        }

        tc_TT_link_status_response.start;

        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(md_CMN_CMN_SlacMmeCmnHeader_001({
                                                 CM_NW_STATS_REQ := '6048'H}),
                                                 md_CMN_CMN_CmNwStatsReq_001()))
                                                 to par_testSystem_plc_node_mac;

        alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_NW_STATS_CNF := '6049'H}),
                                    md_CMN_CMN_CmNwStatsCnf_001())) {

               setverdict(fail,"The SUTs node was detected in the current " &
                               "logical network.");
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_NW_STATS_CNF := '6049'H}),
                                    md_CMN_CMN_CmNwStatsCnf_002())) {
```

```
                setverdict(pass,"The SUTs node has left the current " &
                                "logical network.");
            }
            [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or " &
                                  "content was received.");
            }
            [] tc_TT_link_status_response.timeout {
                setverdict(fail,"CM_NW_STATS timeout.");
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_PLCLinkStatus_009(in HAL_61851_Listener
                                                 v_HAL_61851_Listener)
                                                 runs on SECC_Tester
                                                 return verdicttype {

        var verdicttype v_verdict := pass;
        var hexstring v_Nmk_old;
        var hexstring v_Nid_old;

        v_Nmk_old := vc_Nmk;
        v_Nid_old := vc_Nid;

        sleep(par_CMN_waitForConnectionLoss);

        v_HAL_61851_Listener.stop;
        v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(false));

        // generate new Nid and Nmk
        vc_Nmk := f_randomHexStringGen(32);
        vc_Nid := fx_generateNID(vc_Nmk);
        v_verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(true);
        if (v_verdict == pass) {
            v_verdict := f_SECC_getPLCLinkTermination(par_TP_match_leave,
                                                      fail);
        }

        // set old Nid and Nmk
        if (v_verdict == pass) {
            vc_Nmk := vc_Nmk;
            vc_Nid := vc_Nid;
            v_verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(true);
        }
        if (v_verdict == pass) {
            f_SECC_checkLeavingLogicalNetwork();
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_PLCLinkStatus_010(in HAL_61851_Listener
                                                 v_HAL_61851_Listener)
                                                 runs on SECC_Tester
                                                 return verdicttype  {

        var verdicttype v_verdict := pass;

        v_HAL_61851_Listener.stop;
        v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(true));

        f_SECC_setIsConfirmationFlagDC();
        if(PICS_CMN_CMN_ChargingMode == aC){
            f_SECC_changeValidDutyCycleRange(10,96);
            vc_validDutyCycleLowerBound2 := 10;
            vc_validDutyCycleUpperBound2 := 96;
        }

        // B toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_setState(B,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // C toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_changeValidStateCondition(C);
        f_SECC_setState(C,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // B toggle
    tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
    f_SECC_changeValidStateCondition(B);
    f_SECC_setState(B,v_HAL_61851_Listener);
    tc_TP_EV_vald_state_duration.timeout;

    tc_T_conn_max_comm.start(par_T_conn_max_comm);

    if (getverdict == pass) {
        v_verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                             (par_T_conn_max_comm -
                                              tc_T_conn_max_comm.read),
                                              fail);
    }
    if (getverdict == pass) {
        var Security_TYPE v_security := cc_hexTls;
        if(PICS_CMN_CMN_IdentificationMode == eIM and
           PIXIT_SECC_CMN_TLS == false) {
            v_security := cc_hexTcp;
        }
        v_verdict := f_SECC_CMN_TB_VTB_SDP_001(v_security, fail);
    }

    return getverdict;
}

function f_SECC_CMN_TB_VTB_PLCLinkStatus_011(in HAL_61851_Listener
                                              v_HAL_61851_Listener)
                                              runs on SECC_Tester
                                              return verdicttype {

    var verdicttype v_verdict := pass;

    sleep(par_CMN_waitForConnectionLoss);

    f_SECC_changeValidStateCondition(EorF);
    f_SECC_changeValidDutyCycleRange(0,0);
    f_SECC_setIsConfirmationFlagVoltage();

    // generate new Nid and Nmk
    vc_Nmk := f_randomHexStringGen(32);
    vc_Nid := fx_generateNID(vc_Nmk);
    v_verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(true);
    if (v_verdict == pass) {
        v_verdict := f_SECC_getPLCLinkTermination(par_TP_match_leave,
                                                   fail);
    }

    if (v_verdict == pass) {
        timer statetimer := par_CMN_HAL_Timeout;
        v_verdict := f_SECC_confirmState(EorF, v_HAL_61851_Listener,
                                         statetimer);
    }

    if (getverdict != pass) {
        setverdict(fail, "The SUT did not apply CP State E or F.");
    }
    else {
        tc_T_step_EF.start(par_T_step_EF_min - cc_offset);
        alt {
            [] tc_T_step_EF.timeout {}
            [] a_SECC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
                setverdict
                (fail, "Invalid message type or content was received.");
            }
        }

        f_SECC_changeValidStateCondition(B);
        f_SECC_setIsConfirmationFlagDC();
        vc_validDutyCycleLowerBound1 := 3;
        vc_validDutyCycleUpperBound1 := 7;
        if (PICS_CMN_CMN_ChargingMode == aC) {
            f_SECC_changeValidDutyCycleRange(10,100);
            vc_validDutyCycleLowerBound2 := 10;
            vc_validDutyCycleUpperBound2 := 100;
        } else {
            f_SECC_changeValidDutyCycleRange(100,100);
            vc_validDutyCycleLowerBound2 := 100;
```

```
                vc_validDutyCycleUpperBound2 := 100;
            }

            tc_T_step_EF.start(par_T_step_EF_max -
                              (par_T_step_EF_min - cc_offset));
            alt {
                [] tc_T_step_EF.timeout;
                [] a_SECC_DCDetection(pt_HAL_61851_Internal_Port,
                                      vc_validDutyCycleLowerBound1,
                                      vc_validDutyCycleUpperBound1,
                                      vc_validDutyCycleLowerBound2,
                                      vc_validDutyCycleUpperBound2) {
                   vc_confirmDC := true;
                }
                [] pt_SLAC_Port.receive {
                    setverdict
                    (fail, "Invalid message type or content was received.");
                }
            }
            if(not vc_confirmDC) {
                v_verdict := f_SECC_confirmDutyCycle(v_HAL_61851_Listener,
                                                     par_T_conn_max_comm,
                                                     fail);
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_PLCLinkStatus_012(in HAL_61851_Listener
                                             v_HAL_61851_Listener)
                                             runs on SECC_Tester
                                             return verdicttype  {

        var verdicttype v_verdict := pass;

        v_HAL_61851_Listener.stop;
        v_HAL_61851_Listener.start(f_SECC_HAL61851Listener(true));

        f_SECC_setIsConfirmationFlagDC();
        if(PICS_CMN_CMN_ChargingMode == aC){
            f_SECC_changeValidDutyCycleRange(10,96);
            vc_validDutyCycleLowerBound2 := 10;
            vc_validDutyCycleUpperBound2 := 96;
        }

        // B toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_setState(B,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // C toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_changeValidStateCondition(C);
        f_SECC_setState(C,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;
        // B toggle
        tc_TP_EV_vald_state_duration.start(par_TP_EV_vald_state_duration);
        f_SECC_changeValidStateCondition(B);
        f_SECC_setState(B,v_HAL_61851_Listener);
        tc_TP_EV_vald_state_duration.timeout;

        tc_T_conn_max_comm.start(par_T_conn_max_comm);

        if (getverdict == pass) {
            alt {
                [] tc_T_step_X1.timeout {
                    log("The SUT did not signal B1/B2 transition earlier " &
                        "than 'par_SECC_T_step_X1'.");
                    repeat;
                }
                [] a_SECC_DCDetection(pt_HAL_61851_Internal_Port,
                                      vc_validDutyCycleLowerBound1,
                                      vc_validDutyCycleUpperBound1,
                                      vc_validDutyCycleLowerBound2,
                                      vc_validDutyCycleUpperBound2) {
                   if(tc_T_step_X1.running) {
                    setverdict(fail, "The SUT has signaled B1/B2 transition earlier " &
                                     "than 'par_SECC_T_step_X1'.");
                   }
                }
                [] pt_SLAC_Port.receive {
                    setverdict(fail, "Invalid message type or content " &
                                     "was received.");
                }
                [] tc_T_conn_max_comm.timeout {
                    setverdict(fail, "The SUT could not signal the " &
                                     "corresponding duty cycle.");
                }
            }
        }
        return getverdict;
    }

    function f_SECC_AC_TB_VTB_PLCLinkStatus_001() runs on SECC_Tester
                                              return verdicttype {
        timer t1;
        var float v_waitForEim := 1.0;

        while(not vc_eimDone and getverdict == pass) {
            log("Wait for user interaction.");
            t1.start(v_waitForEim);
            alt {
                [] t1.timeout;
            }
        }
        if(getverdict == pass) {
            sleep(par_SECC_change_to_Nominal);
            f_SECC_getDcNominalStatus();
        }
        return getverdict;
    }

    function f_SECC_AC_TB_VTB_PLCLinkStatus_002(in HAL_61851_Listener
                                            v_HAL_61851_Listener)
                                            runs on SECC_Tester
                                            return verdicttype {

        var verdicttype v_verdict := pass;

        sleep(par_CMN_waitForConnectionLoss);

        // generate new Nid and Nmk
        vc_Nmk := f_randomHexStringGen(32);
        vc_Nid := fx_generateNID(vc_Nmk);
        v_verdict := f_SECC_CMN_TB_VTB_CmSetKey_001(true);
        if (v_verdict == pass) {
            v_verdict := f_SECC_getPLCLinkTermination(par_TP_match_leave,
                                                      fail);
        }

        if (getverdict == pass) {
            f_SECC_CMN_TB_VTB_CmSlacParm_001(fail);
        }
        return getverdict;
    }
}
```

E.1.7 SECC functions for CmAmpMap

```
module TestBehavior_SECC_CmAmpMap {

    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Templates_CMN_CmAmpMap all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from TestBehavior_SECC_CommonBehavior all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from Services_HAL_61851 all;
    import from DataStructure_HAL_61851 all;
    import from TTlibrary_Logging all;
    import from TestBehavior_SECC_SDP all;
    import from DataStructure_SDP all;
    import from Pics_15118 all;
    import from Pixit_15118_2 all;
```

```
 function f_SECC_CMN_TB_VTB_CmAmpMap_001(in verdicttype v_vct)
                                        runs on SECC_Tester
                                        return verdicttype {

    var boolean v_repetition := true;
    var integer v_counter := 0;

    while(v_repetition){

        tc_TT_match_response.start(par_TT_match_response);
        v_counter := v_counter + 1;
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_AMP_MAP_REQ := '601C'H}),
                          m_CMN_CMN_CmAmpMapReq_001()))
                          to vc_sut_mac;

         alt {
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_AMP_MAP_CNF := '601D'H}),
                                       md_CMN_CMN_CmAmpMapCnf_001('00'H))) {

                    setverdict(pass,"CM_AMP_MAP.CNF is correct.");
                    v_repetition := false;
                    tc_TT_match_response.stop;
                }
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_AMP_MAP_CNF := '601D'H}),
                                       md_CMN_CMN_CmAmpMapCnf_001('01'H))) {

                    setverdict(v_vct,"The SUT could not perform the " &
                                     "Amplitude map exchange.");
                    v_repetition := false;
                    tc_TT_match_response.stop;
                }
                [] pt_SLAC_Port.receive {
                   setverdict(v_vct, "Invalid message type or content " &
                                    "was received.");
                   v_repetition := false;
                   tc_TT_match_response.stop;
                }
                [] tc_TT_match_response.timeout {
                    log("TT_match_response timeout.");
                    if(v_counter mod (par_C_EV_match_retry+1) == 0){
                       setverdict(v_vct,"The SUT did not response to the " &
                                     "CmAmpMapReq message.");
                        v_repetition := false;
                    } else {
                        log("A new CM_AMP_MAP.REQ message will be sent.");
                    }
                }
         }
    }
    return getverdict;
}

function f_SECC_CMN_TB_VTB_CmAmpMap_002(in verdicttype v_vct)
                                           runs on SECC_Tester
                                           return verdicttype {

   var MME v_requestMessage;

   tc_TT_amp_map_exchange.start(par_TT_amp_map_exchange);

   alt {
           []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_AMP_MAP_REQ := '601C'H}),
                                  md_CMN_CMN_CmAmpMapReq_002(?,?)))
                                  -> value v_requestMessage {

               tc_TT_match_response.start(par_TT_match_response);
               pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_AMP_MAP_CNF := '601D'H}),
                                 md_CMN_CMN_CmAmpMapCnf_001('00'H)))
```

```
                              to vc_sut_mac;

                var Amlen_TYPE v_amlen := v_requestMessage.mme_payload.
                                              payload.cm_amp_map_req.amlen;
                var ListofAmdata_TYPE v_listAmdata := v_requestMessage.mme_payload.
                                                    payload.cm_amp_map_req.listAmdata;

                setverdict(pass,"CM_AMP_MAP.REQ is correct.");
                tc_TT_amp_map_exchange.stop;
            }
            [] pt_SLAC_Port.receive {
                setverdict(v_vct, "Invalid message type or content was received.");
            }
            [] tc_TT_amp_map_exchange.timeout {
                setverdict(v_vct,"TT_amp_map_exchange timeout. " &
                                   "No Amplitude Map exchange was performed by the SUT.");
            }
        }
        return getverdict;
    }

    function f_SECC_CMN_TB_VTB_CmAmpMap_003() runs on SECC_Tester return verdicttype {

         var integer v_count := 0;
         tc_TT_amp_map_exchange.start(par_TT_amp_map_exchange);

         alt {
             []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_AMP_MAP_REQ := '601C'H}),
                                   md_CMN_CMN_CmAmpMapReq_002(?,?))) {

                 if(v_count > 0){
                     setverdict(pass,"CM_AMP_MAP.REQ message was repeated.",v_count);
                 } else { tc_TT_amp_map_exchange.stop;}

                 v_count := v_count + 1;
                 tc_TT_match_response.start(par_TT_match_response);

                 if(v_count > par_C_EV_match_retry) {

                     alt{
                         []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_AMP_MAP_REQ := '601C'H}),
                                               md_CMN_CMN_CmAmpMapReq_002(?,?))) {

                           setverdict(fail,"CM_AMP_MAP.REQ message was repeated, " &
                                           "but v_count > par_C_EV_match_retry.");
                         }
                         [] pt_SLAC_Port.receive {
                            setverdict(fail, "Invalid message type or content " &
                                             "was received.");
                         }
                         [] tc_TT_match_response.timeout {
                           setverdict(pass,"TT_match_response timeout. " &
                                           "The total number of retries is reached, " &
                                           "the Matching process " &
                                           "shall be considered as FAILED.");
                         }
                       }
                 }
                 else{
                     repeat;
                 }
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
             }
             [] tc_TT_amp_map_exchange.timeout {
                 setverdict(fail,"No Amplitude Map exchange was performed by the SUT.");
             }
             [] tc_TT_match_response.timeout {
                 setverdict(fail,"The SUT did not retransmit the " &
                                 "CM_AMP_MAP.REQ message.");
             }
         }
        return getverdict;
```

```
     }

    function f_SECC_CMN_TB_VTB_CmAmpMap_004() runs on SECC_Tester return verdicttype {

         var integer v_count := 0;
         tc_TT_amp_map_exchange.start(par_TT_amp_map_exchange);

         alt {
             []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_AMP_MAP_REQ := '601C'H}),
                                    md_CMN_CMN_CmAmpMapReq_002(?,?))) {

                 if(v_count > 0){
                     setverdict(pass,"CM_AMP_MAP.REQ message was repeated.",v_count);
                 } else { tc_TT_amp_map_exchange.stop;}

                 v_count := v_count + 1;
                 tc_TT_match_response.start(par_TT_match_response);
                 // send invalid CM_AMP_MAP.CNF message
                 pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_AMP_MAP_CNF := '601D'H}),
                                   md_CMN_CMN_CmAmpMapCnf_001('FF'H)))
                                   to vc_sut_mac;

                 if(v_count > par_C_EV_match_retry) {

                     alt{
                         []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                                CM_AMP_MAP_REQ := '601C'H}),
                                                md_CMN_CMN_CmAmpMapReq_002(?,?))) {

                           setverdict(fail,"CM_AMP_MAP.REQ message was repeated, " &
                                           "but v_count > par_C_EV_match_retry.");
                         }
                         [] pt_SLAC_Port.receive {
                            setverdict(fail, "Invalid message type or content " &
                                             "was received.");
                         }
                         [] tc_TT_match_response.timeout {
                           setverdict(pass,"TT_match_response timeout. " &
                                           "The total number of retries is reached, " &
                                           "the Matching process " &
                                           "shall be considered as FAILED.");
                         }
                       }
                 }
                 else{
                     repeat;
                 }
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
             }
             [] tc_TT_amp_map_exchange.timeout {
                 setverdict(fail,"No Amplitude Map exchange was performed by the SUT.");
             }
             [] tc_TT_match_response.timeout {
                 setverdict(fail,"The SUT did not retransmit the " &
                                 "CM_AMP_MAP.REQ message.");
             }
          }
          return getverdict;
    }

    function f_SECC_CMN_TB_VTB_CmAmpMap_005() runs on SECC_Tester return verdicttype {

       var boolean v_repetition := true;
       var integer v_counter := 0;

       while(v_repetition){

           tc_TT_match_response.start(par_TT_match_response);
           v_counter := v_counter + 1;
           // send invalid CM_AMP_MAP.REQ message
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
```

```
                        CM_AMP_MAP_REQ := '601C'H}),
                        md_CMN_CMN_CmAmpMapReq_003('0000'H)))
                        to vc_sut_mac;

         alt {
             []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_AMP_MAP_CNF := '601D'H}),
                                 md_CMN_CMN_CmAmpMapCnf_001(?))) {

                 setverdict(fail,"Received CM_AMP_MAP.CNF message " &
                                 "was not expected.");
                 v_repetition := false;
                 tc_TT_match_response.stop;
             }
             [] pt_SLAC_Port.receive {

                setverdict(fail, "Invalid message type or content " &
                                 "was received.");
                v_repetition := false;
                tc_TT_match_response.stop;
             }
             [] tc_TT_match_response.timeout {
                 log("TT_match_response timeout.");
                 setverdict(pass,"The SUT did not response to the " &
                                 "invalid CM_AMP_MAP.REQ message.");
                 if(v_counter mod (par_C_EV_match_retry+1) == 0){
                    v_repetition := false;
                 } else {
                    log("A new invalid CM_AMP_MAP.REQ message will be sent.");
                 }
             }
         }
     }
     return getverdict;
}

function f_SECC_CMN_TB_VTB_CmAmpMap_006() runs on SECC_Tester return verdicttype {

   var boolean v_repetition := true;
   var integer v_counter := 0;

   while(v_repetition){

       tc_TT_match_response.start(par_TT_match_response);
       v_counter := v_counter + 1;
       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                         md_CMN_CMN_SlacMmeCmnHeader_001({
                         CM_AMP_MAP_REQ := '601C'H}),
                         m_CMN_CMN_CmAmpMapReq_001()))
                         to vc_sut_mac;

        alt {
               []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_AMP_MAP_CNF := '601D'H}),
                                   md_CMN_CMN_CmAmpMapCnf_001('00'H))) {

                   setverdict(pass,"CM_AMP_MAP.CNF is correct.");
                   tc_TT_match_response.stop;
                   if(v_counter > 1) {
                       v_repetition := false;
                   }
               }
               []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_AMP_MAP_CNF := '601D'H}),
                                   md_CMN_CMN_CmAmpMapCnf_001('01'H))) {

                   setverdict(fail,"The SUT could not perform the " &
                                   "Amplitude map exchange.");
                   v_repetition := false;
                   tc_TT_match_response.stop;
               }
               [] pt_SLAC_Port.receive {

                  setverdict(fail, "Invalid message type or content " &
                                  "was received.");
                  v_repetition := false;
                  tc_TT_match_response.stop;
               }
               [] tc_TT_match_response.timeout {
                  setverdict(fail,"The SUT did not response to the " &
                                  "CM_AMP_MAP.REQ message.");
                  v_repetition := false;
               }
          }
      }
      return getverdict;
   }

   function f_SECC_CMN_TB_VTB_CmAmpMap_007() runs on SECC_Tester return verdicttype {

      var integer v_counter := 0;

      tc_TT_match_response.start(par_TT_match_response);
      for (var integer i:=0; i<3; i:=i + 1) {
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_AMP_MAP_REQ := '601C'H}),
                          m_CMN_CMN_CmAmpMapReq_001()))
                          to vc_sut_mac;
      }

      alt {
          []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_AMP_MAP_CNF := '601D'H}),
                                 md_CMN_CMN_CmAmpMapCnf_001('00'H))) {

              setverdict(pass,"CM_AMP_MAP.CNF is correct.");
              v_counter := v_counter + 1;
              tc_TT_match_response.stop;
              tc_TT_match_response.start(par_TT_match_response);
              if(v_counter < 3) {
                  repeat;
              }
          }
          []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_AMP_MAP_CNF := '601D'H}),
                                 md_CMN_CMN_CmAmpMapCnf_001('01'H))) {

              setverdict(fail,"The SUT could not perform the " &
                              "Amplitude map exchange.");
              tc_TT_match_response.stop;
          }
          [] pt_SLAC_Port.receive {

             setverdict(fail, "Invalid message type or content " &
                              "was received.");
             tc_TT_match_response.stop;
          }
          [] tc_TT_match_response.timeout {
             setverdict(fail,"The SUT did not response to the " &
                             "CM_AMP_MAP.REQ message.");
          }
       }
      return getverdict;
   }

   function f_SECC_CMN_TB_VTB_CmAmpMap_008() runs on SECC_Tester return verdicttype {

      var Security_TYPE v_security := cc_hexTls;
      if(PICS_CMN_CMN_IdentificationMode == eIM and
         PIXIT_SECC_CMN_TLS == false) {
            v_security := cc_hexTcp;
      }
      f_SECC_CMN_TB_VTB_SDP_001(v_security, fail);
      return getverdict;
   }
}
```

E.2 EVCC + PLC bridge functions

This subclause includes all functions *specifications* where the EV is defined as SUT.

E.2.1 EVCC functions for CmSlacParm

```
module TestBehavior_EVCC_CmSlacParm {

    import from Templates_CMN_CmSlacParm all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmStartAttenCharInd all;
    import from Templates_CMN_CmSlacParm all;
    import from Templates_SECC_CmAttenCharInd all;
    import from Templates_CMN_CmAttenCharRsp all;
    import from Templates_CMN_CmMnbcSoundInd all;
    import from Templates_CMN_CmSlacMatch all;
    import from Templates_CMN_CmValidate all;
    import from ComponentsAndPorts all;
    import from Timer_15118_3 all;
    import from Pixit_15118_3 all;
    import from Pics_15118_3 all;
    import from DataStructure_SLAC all;
    import from LibFunctions_15118_3  { group generalFunctions; };
    import from Services_HAL_61851 all;
    import from Services_PLCLinkStatus all;
    import from Timer_15118 all;

    function f_EVCC_CMN_TB_VTB_CmSlacParm_001(in verdicttype v_vct)
                                          runs on EVCC_Tester
                                          return verdicttype {

        var MME v_responseMessage;
        var MACAddress_TYPE v_address;

        alt{
             []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_PARM_REQ := '6064'H}),
                                    md_CMN_CMN_CmSlacParmReq_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                    -> value v_responseMessage
                                    sender vc_sut_mac {

                setverdict(pass,"CM_SLAC_PARM.REQ is correct.");
                vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
             }

             [] a_EVCC_processPLCLinkNotifications_001();
             [] pt_SLAC_Port.receive {
                setverdict(v_vct, "Invalid message type or content was received.");
             }
             [] tc_TT_EVSE_SLAC_init.timeout {
                setverdict(v_vct,"TT_EVSE_SLAC_init timeout. SECC assumes that no SLAC " &
                                 "will be performed.");
             }
         }
         return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmSlacParm_002() runs on EVCC_Tester return verdicttype {

        var MME v_responseMessage;
        var MACAddress_TYPE v_address;
        var integer v_count := 0;

        alt{
             []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_PARM_REQ := '6064'H}),
                                    md_CMN_CMN_CmSlacParmReq_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                    -> value v_responseMessage sender vc_sut_mac {

                vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
                if(v_count > 0){
                    setverdict(pass,"CM_SLAC_PARM.REQ message was repeated.",v_count);
                }
                v_count := v_count + 1;
```

```
            tc_TT_match_sequence.start(par_TT_match_sequence);

            if(v_count > par_C_EV_match_retry) {
              tc_TT_match_sequence.stop;
              tc_TT_match_response.start(par_TT_match_response +
                                         (2.0 * par_CMN_Transmission_Delay));
              alt{
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SLAC_PARM_REQ := '6064'H}),
                                       md_CMN_CMN_CmSlacParmReq_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                       -> value v_responseMessage sender vc_sut_mac {

                  setverdict(fail,"CM_SLAC_PARM.REQ message was repeated, but v_count > " &
                                   "par_C_EV_match_retry.");
                }
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                }
                [] tc_TT_match_response.timeout {
                  setverdict(pass,"TT_match_response timeout. " &
                                   "The total number of retries is reached, " &
                                   "the Matching process " &
                                   "shall be considered as FAILED.");
                }
              }
            }
            else{
                repeat;
            }
         }
         [] a_EVCC_processPLCLinkNotifications_001();
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
         }
         [] tc_TT_EVSE_SLAC_init.timeout {
            setverdict(fail,"TT_EVSE_SLAC_init timeout. SECC assumes " &
                            "that no SLAC will be performed.");
         }
         [] tc_TT_match_sequence.timeout {
           setverdict(fail,"TT_match_sequence timeout. " &
                             "CM_SLAC_PARM.REQ message was not repeated.");
         }
     }
     return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmSlacParm_003(in template(present) MME_Payload invalidPayload)
                                         runs on EVCC_Tester return verdicttype {

    var MME v_responseMessage;
    var MACAddress_TYPE v_address;
    var integer v_count := 0;

    alt{
         []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_PARM_REQ := '6064'H}),
                                md_CMN_CMN_CmSlacParmReq_001(
                                m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                -> value v_responseMessage sender vc_sut_mac {

            vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
            if(not(ispresent(invalidPayload.payload.cm_slac_parm_cnf.runid))) {
                invalidPayload.payload.cm_slac_parm_cnf.runid := vc_RunID;
            }
            if(not(ispresent(invalidPayload.payload.cm_slac_parm_cnf.forwarding_sta))) {
                invalidPayload.payload.cm_slac_parm_cnf.forwarding_sta := vc_sut_mac;
            }

            if(v_count > 0){
                setverdict(pass,"CM_SLAC_PARM.REQ message was repeated.",v_count);
            }
            v_count := v_count + 1;
            tc_TT_match_sequence.start(par_TT_match_sequence);
            // send invalid CM_SLAC_PARM.CNF message
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
```

```
                          CM_SLAC_PARM_CNF := '6065'H}), invalidPayload))
                          to vc_sut_mac;

               if(v_count > par_C_EV_match_retry) {
                 tc_TT_match_sequence.stop;
                 tc_TT_match_response.start(par_TT_match_response +
                                      (2.0 * par_CMN_Transmission_Delay));
                 alt{
                   []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SLAC_PARM_REQ := '6064'H}),
                                       md_CMN_CMN_CmSlacParmReq_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                       -> value v_responseMessage sender vc_sut_mac {

                     setverdict(fail,"CM_SLAC_PARM.REQ message was repeated, but v_count > " &
                                  "par_C_EV_match_retry.");
                   }
                   [] pt_SLAC_Port.receive {
                      setverdict(fail, "Invalid message type or content was received.");
                   }
                   [] tc_TT_match_response.timeout {
                     setverdict(pass,"TT_match_response timeout. " &
                                  "The total number of retries is reached, " &
                                  "the Matching process " &
                                  "shall be considered as FAILED.");
                   }
                 }
               }
               else{
                   repeat;
               }
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_START_ATTEN_CHAR_IND := '606A'H}),
                                md_CMN_CMN_CmStartAttenCharInd_001(
                                m_CMN_CMN_SlacPayloadHeader_001(), ?, ?, '01'H, ?, vc_RunID)))
                                -> value v_responseMessage {

                setverdict(fail,"CM_START_ATTEN_CHAR.IND message was received but not expected.");
            }
            [] a_EVCC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_EVSE_SLAC_init.timeout {
               setverdict(fail,"TT_EVSE_SLAC_init timeout. SECC assumes " &
                               "that no SLAC will be performed.");
            }
            [] tc_TT_match_sequence.timeout {
              setverdict(fail,"TT_match_sequence timeout. " &
                                 "CM_SLAC_PARM.REQ message was not repeated.");
            }
        }
        return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmSlacParm_004() runs on EVCC_Tester return verdicttype {

        var MME v_responseMessage;
        var MACAddress_TYPE v_address;
        var boolean v_isInvalidMes := false;
        var SourceRnd_Type v_source_rnd := f_randomHexStringGen(32);
        timer t;

        alt{
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_PARM_REQ := '6064'H}),
                                md_CMN_CMN_CmSlacParmReq_001(
                                m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                -> value v_responseMessage sender vc_sut_mac {

               setverdict(pass,"CM_SLAC_PARM.REQ is correct.");
               t.start(par_TT_EV_atten_results);
               vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
```

```
// send CM_SLAC_PARM.REQ message
pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                  md_CMN_CMN_SlacMmeCmnHeader_001({
                  CM_SLAC_PARM_REQ := '6064'H}),
                  md_CMN_CMN_CmSlacParmReq_001(
                  m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                  to vc_sut_mac;

// send CM_START_ATTEN_CHAR.IND message
pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                  md_CMN_CMN_SlacMmeCmnHeader_001({
                  CM_START_ATTEN_CHAR_IND := '606A'H}),
                  md_CMN_CMN_CmStartAttenCharInd_001(
                  m_CMN_CMN_SlacPayloadHeader_001(), '0A'H,
                  '06'H, '01'H, par_testSystem_mac, vc_RunID)))
                  to vc_sut_mac;

// send CM_MNBC_SOUND.IND message
pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                  md_CMN_CMN_SlacMmeCmnHeader_001({
                  CM_MNBC_SOUND_IND := '6076'H}),
                  md_CMN_CMN_CmMnbcSoundInd_001(
                  m_CMN_CMN_SlacPayloadHeader_001(),
                  int2hex(10,2), vc_RunID, v_source_rnd)))
                  to vc_sut_mac;

// send CM_ATTEN_CHAR.RSP message
pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                  md_CMN_CMN_SlacMmeCmnHeader_001({
                  CM_ATTEN_CHAR_RSP :='606F'H}),
                  md_CMN_CMN_CmAttenCharRsp_001(
                  m_CMN_CMN_SlacPayloadHeader_001(),
                  md_CMN_CMN_Acvarfield_001(par_testSystem_mac, vc_RunID))))
                  to vc_sut_mac;

// send CM_VALIDATE.REQ message
pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                  md_CMN_CMN_SlacMmeCmnHeader_001({
                  CM_VALIDATE_REQ := '6078'H}),
                  m_CMN_CMN_CmValidateReq_001()))
                  to vc_sut_mac;

// send message
pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                  md_CMN_CMN_SlacMmeCmnHeader_001({
                  CM_SLAC_MATCH_REQ := '607C'H}),
                  md_CMN_CMN_CmSlacMatchReq_001(
                  m_CMN_CMN_SlacPayloadHeader_001(), par_testSystem_mac,
                  vc_sut_mac, vc_RunID)))
                  to vc_sut_mac;

alt {
    []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                           md_CMN_CMN_SlacMmeCmnHeader_001({
                           CM_SLAC_PARM_REQ := '6064'H}),
                           md_CMN_CMN_CmSlacParmReq_001(
                           m_CMN_CMN_SlacPayloadHeader_001(), ?))) {
        repeat;
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_SLAC_PARM_CNF := '6065'H}),?)) {
        v_isInvalidMes := true;
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_ATTEN_CHAR_IND := '606E'H}),?)) {
        v_isInvalidMes := true;
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_VALIDATE_CNF := '6079'H}),?)) {
        v_isInvalidMes := true;
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_SLAC_MATCH_CNF := '607D'H}),?)) {
        v_isInvalidMes := true;
    }
```

```
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_SLAC_PARM_CNF := '6065'H}),?)) {
                     v_isInvalidMes := true;
                  }
                  [] pt_SLAC_Port.receive {
                     setverdict(fail, "Invalid message type or content was received.");
                  }
                  [] t.timeout {
                     setverdict(pass,"The SUT did not respond to the EVCC messages.");
                  }
              }
              if (v_isInvalidMes) {
                  setverdict(fail, "The EVCC shall not respond to the following messages: " &
                                   "CM_SLAC_PARM.REQ, CM_START_ATTEN_CHAR.IND, " &
                                   "CM_MNBC_SOUND.IND, CM_ATTEN_CHAR.RSP, CM_VALIDATE.REQ, " &
                                   "CM_SLAC_MATCH.REQ.");
              }
           }
           [] a_EVCC_processPLCLinkNotifications_001();
           [] pt_SLAC_Port.receive {
              setverdict(fail, "Invalid message type or content was received.");
           }
           [] tc_TT_EVSE_SLAC_init.timeout {
              setverdict(fail,"TT_EVSE_SLAC_init timeout. SECC assumes " &
                              "that no SLAC will be performed.");
           }
       }
       return getverdict;
   }

   function f_EVCC_CMN_TB_VTB_CmSlacParm_005() runs on EVCC_Tester return verdicttype {

       var MME v_responseMessage;
       var MACAddress_TYPE v_address;
       var integer v_count := 0;

       tc_TP_matching_repetition.start(par_TT_matching_repetition);

       alt{
            []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_SLAC_PARM_REQ := '6064'H}),
                                   md_CMN_CMN_CmSlacParmReq_001(
                                   m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                   -> value v_responseMessage sender vc_sut_mac {

              vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
              if(v_count > 0){
                  setverdict(pass,"CM_SLAC_PARM.REQ message was repeated.",v_count);
              }
              tc_TT_match_sequence.start(par_TT_match_sequence);
              v_count := v_count + 1;

              if(v_count > par_C_EV_match_retry) {
                tc_TT_match_sequence.stop;
                tc_TT_match_response.start(par_TT_match_response +
                                           (2.0 * par_CMN_Transmission_Delay));
                alt{
                  []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_SLAC_PARM_REQ := '6064'H}),
                                         md_CMN_CMN_CmSlacParmReq_001(
                                         m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                         -> value v_responseMessage sender vc_sut_mac {

                    setverdict(fail,"CM_SLAC_PARM.REQ message was repeated, but v_count > " &
                                    "par_C_EV_match_retry.");
                  }
                  [] pt_SLAC_Port.receive {
                     setverdict(fail, "Invalid message type or content was received.");
                  }
                  [] tc_TT_match_response.timeout {
                     setverdict(pass,"TT_match_response timeout. " &
                                       "The total number of retries is reached, " &
                                    "the Matching process " &
                                    "shall be considered as FAILED.");
                    tc_TP_matching_rate.start(PIXIT_EVCC_CMN_TTMatchingRate +
                                              par_CMN_Transmission_Delay);
                    if(tc_TP_matching_repetition.running) {
                       v_count := 0;
                        repeat;
                    }
                    else {
                     alt{
                       []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                              md_CMN_CMN_SlacMmeCmnHeader_001({
                                              CM_SLAC_PARM_REQ := '6064'H}),
                                              md_CMN_CMN_CmSlacParmReq_001(
                                              m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                              -> value v_responseMessage sender vc_sut_mac {

                         setverdict(fail,"CM_SLAC_PARM.REQ message was repeated, but " &
                                         "tc_TP_matching_repetition timer has expired.");
                       }
                       [] pt_SLAC_Port.receive {
                          setverdict(fail, "Invalid message type or content was received.");
                       }
                       [] tc_TP_matching_rate.timeout {
                         setverdict(pass,"SUT did not start a new matching process after" &
                                         "tc_TP_matching_repetition timeout.");
                       }
                     }
                   }
                 }
               }
             }
             else{
                 repeat;
             }
          }
          [] a_EVCC_processPLCLinkNotifications_001();
          [] pt_SLAC_Port.receive {
             setverdict(fail, "Invalid message type or content was received.");
          }
          [] tc_TT_EVSE_SLAC_init.timeout {
             setverdict(fail,"TT_EVSE_SLAC_init timeout. SECC assumes " &
                             "that no SLAC will be performed.");
          }
          [] tc_TT_match_sequence.timeout {
            setverdict(fail,"TT_match_sequence timeout. " &
                              "CM_SLAC_PARM.REQ message was not repeated.");
          }
          [] tc_TP_matching_rate.timeout {
            setverdict(fail,"TP_matching_rate timeout. " &
                              "SUT did not start a new matching process.");
          }
        }
        return getverdict;
    }
}
```

E.2.2 EVCC functions for AttenuationCharacterization

```
module TestBehavior_EVCC_AttenuationCharacterization {

    import from Templates_CMN_CmStartAttenCharInd all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmMnbcSoundInd all;
    import from Templates_CMN_CmAttenCharRsp all;
    import from Templates_EVCC_CmAttenCharInd all;
    import from Templates_EVCC_CmAttenProfileInd all;
    import from Templates_CMN_CmSlacParm all;
    import from ComponentsAndPorts all;
    import from Pics_15118 all;
    import from Pics_15118_3 all;
    import from Pixit_15118_3 all;
    import from Timer_15118_3 all;
    import from Services_HAL_61851 all;
    import from DataStructure_SLAC all;
    import from LibFunctions_15118_3 { group generalFunctions; }
    import from Services_PLCLinkStatus all;
    import from DataStructure_HAL_61851 all;
```

```
function averageCalc(in ResponseMessageList_TYPE resMessagelist, in integer vcount)
                     return AttenProfile_TYPE {

    var AttenProfile_TYPE attenuation_list;
    var integer avg :=0;

    for (var integer i:=0; i<58; i:=i+1)
    {
        for (var integer j:=0;j<vcount;j:=j+1)
        {
            avg:=avg+hex2int(resMessagelist[j].mme_payload.payload.cm_atten_profile_ind.
                                 attenuation_list.attenuation[i]);
        }
        avg:=avg/vcount;
        attenuation_list.attenuation[i]:=int2hex(avg,2);
    }
    return attenuation_list;
}

function f_EVCC_CMN_TB_VTB_AttenuationCharacterization_001(in verdicttype v_vct)
                                                            runs on EVCC_Tester
                                                            return verdicttype {

        var MME v_responseMessage;
       var integer v_Num_soundsInt;
       var ResponseMessageList_TYPE reponseMessageList;
       var hexstring  v_variable;
       var integer v_count := 0;
       var boolean v_isRunning := true;
       var boolean v_repetition := true;
       var AttenProfile_TYPE v_attenuation_list;
       var integer v_count2 := 0;
       var integer v_countDecrement := -1;
       var integer v_countStart := -1;
       var integer v_countStop := -1;
       var boolean v_firstSound := true;

       tc_TT_match_sequence.start(par_TT_match_sequence);
       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_SLAC_PARM_CNF := '6065'H}),
                         md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                         m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                         to vc_sut_mac;

       alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_START_ATTEN_CHAR_IND := '606A'H}),
                                    md_CMN_CMN_CmStartAttenCharInd_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(), ?, ?,
                                    '01'H, ?, vc_RunID)))
                                    -> value v_responseMessage {

                if(v_count2 == 0) {
                    tc_TT_EVSE_match_MNBC.start(par_TT_EVSE_match_MNBC);
                    tc_TT_match_sequence.stop;

                    vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                     cm_start_atten_char_ind.num_sounds;
                    v_Num_soundsInt := hex2int(vc_Num_sounds);
                }
                setverdict(pass,"CM_START_ATTEN_CHAR.IND is correct.");
                v_count2 := v_count2 + 1;
                if(v_count2 < cc_numberOfStartAtten) {
                    repeat;
                }
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_SLAC_PARM_REQ := '6064'H}),
                                    md_CMN_CMN_CmSlacParmReq_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

               tc_TT_match_sequence.start(par_TT_match_sequence);
               pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_SLAC_PARM_CNF := '6065'H}),
                        md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                        m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                        to vc_sut_mac;

             log("A further CM_SLAC_PARM.REQ message was received. " &
                    "A new CM_SLAC_PARM.CNF has to be send.");
             repeat;
          }
          [] pt_SLAC_Port.receive {
             setverdict(v_vct, "Invalid message type or content was received.");
             if(v_count2 < cc_numberOfStartAtten) {
                    setverdict(v_vct, "A wrong number of CM_START_ATTEN_CHAR.IND " &
                                         "message was received.");
             }
          }
          [] tc_TT_match_sequence.timeout {
              setverdict(v_vct,"TT_match_sequence timeout. " &
                        "No CM_START_ATTEN_CHAR.IND " &
                        "message was received. Matching process shall be " &
                        "considered as FAILED.");
              break;
          }
          [] tc_TT_EVSE_match_MNBC.timeout {
              setverdict(v_vct,"TT_EVSE_match_MNBC timeout. A wrong number of " &
                        "CM_START_ATTEN_CHAR.IND message was received. " &
                        "Matching process shall be considered as FAILED.");
          }
      }

     if(getverdict == pass) {
         while (v_isRunning) {
             alt {
                   [v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_MNBC_SOUND_IND := '6076'H}),
                                        md_CMN_CMN_CmMnbcSoundInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        ?, vc_RunID, ?)))
                                        -> value v_responseMessage {

                      v_countStart := hex2int(v_responseMessage.
                                         mme_payload.payload.
                                         cm_mnbc_sound_ind.count);
                      v_firstSound := false;
                      if(v_countStart == cc_numberOfSoundings) {
                          v_countDecrement := v_countStart - 1;
                          v_countStop := 1;
                      } else if(v_countStart == cc_numberOfSoundings - 1){
                          v_countDecrement := v_countStart - 1;
                          v_countStop := 0;
                      }
                      else {
                              setverdict(v_vct, "The field 'count' has an " &
                                                  "invalid value.");
                              v_isRunning := false;
                              break;
                      }
                      repeat;
                   }
                   [not v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_MNBC_SOUND_IND := '6076'H}),
                                       md_CMN_CMN_CmMnbcSoundInd_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(),
                                       int2hex(v_countDecrement,2),
                                       vc_RunID, ?))) {
                      v_countDecrement := v_countDecrement - 1;
                      repeat;
                   }
                   [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_ATTEN_PROFILE_IND := '6086'H}),
                                        md_EVCC_CMN_CmAttenProfileInd_001(
                                        vc_sut_mac, ?, *)))
                                        -> value v_responseMessage {

                      if(ispresent(v_responseMessage.mme_payload.payload.
                                  cm_atten_profile_ind.attenuation_list)) {
```

```
                        if(v_responseMessage.mme_payload.payload.
                           cm_atten_profile_ind.num_groups != '3A'H) {
                            setverdict(v_vct,"Invalid numGroups value detected.");
                            v_isRunning := false;
                            break;
                        }
                        reponseMessageList[v_count] := v_responseMessage;
                        v_count := v_count + 1;
                    }
                    else {
                        log("Attenuation list was empty, the received message could not " &
                            "be considered for attenuation calculation.");
                    }
                 }
                 [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_SLAC_PARM_REQ := '6064'H}),
                                         md_CMN_CMN_CmSlacParmReq_001(
                                         m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

                    setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                      "New Matching process is started.");
                    v_isRunning := false;
                 }
                 [] pt_SLAC_Port.receive {
                    setverdict(v_vct, "Invalid message type or " &
                                      "content was received.");
                    v_isRunning := false;
                 }
                 [] tc_TT_EVSE_match_MNBC.timeout {
                    v_isRunning := false;
                  }
               }
           if(v_count == v_Num_soundsInt){
               tc_TT_EVSE_match_MNBC.stop;
               v_isRunning := false;
           }
        }
    }
    if (v_count>0){
           if(v_countDecrement != (v_countStop - 1)) {
           setverdict(v_vct,"A wrong number of CM_MNBC_SOUND.IND messages " &
                            "was received.");
           } else {
           vc_Num_sounds := int2hex(v_count,2);
           setverdict(pass,"CM_MNBC_SOUND.IND is correct.");
           setverdict(pass,"CM_ATTEN_PROFILE.IND is correct.");
           if(PIXIT_EVCC_CMN_CmValidate == cmValidate) {
              v_attenuation_list := m_EVCC_CMN_atten_list_002();
           } else {
               v_attenuation_list := averageCalc(reponseMessageList, v_count);
           }
           }
    }
    else {
        setverdict(v_vct,"No Atten Profile messages received.");
    }

    if(getverdict == pass) {
        v_count := 0;
        while(v_repetition){

            tc_TT_match_response.start(par_TT_match_response);
            v_count := v_count + 1;
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_ATTEN_CHAR_IND := '606E'H}),
                              md_EVCC_CMN_CmAttenCharInd_001(
                              m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                              vc_RunID, vc_Num_sounds, v_attenuation_list)))
                              to vc_sut_mac;
             alt {
                   [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                             md_CMN_CMN_SlacMmeCmnHeader_001({
                                             CM_ATTEN_CHAR_RSP :='606F'H}),
                                              md_CMN_CMN_CmAttenCharRsp_001(
                                              m_CMN_CMN_SlacPayloadHeader_001(),
                                              md_CMN_CMN_Acvarfield_001(
                                        vc_sut_mac, vc_RunID)))) {

                         setverdict(pass,"CM_ATTEN_CHAR.RSP is correct.");
                         v_repetition := false;
                         tc_TT_match_response.stop;
                         tc_TT_EVSE_match_session.start(par_TT_EVSE_match_session);
                     }
                     [v_gracefulHandling] pt_SLAC_Port.receive(
                                             md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_ATTEN_CHAR_RSP :='606F'H}),?)) {

                         log("The CM_ATTEN_CHAR.RSP message content is " &
                             "not conform but graceful message handling " &
                             "is enabled.");
                         v_repetition := false;
                         tc_TT_match_response.stop;
                         tc_TT_EVSE_match_session.start(par_TT_EVSE_match_session);
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_SLAC_PARM_REQ := '6064'H}),
                                             md_CMN_CMN_CmSlacParmReq_001(
                                             m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                         setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                           "New Matching process is started.");
                         v_repetition := false;
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_MNBC_SOUND_IND := '6076'H}),?)) {
                        // CM_ATTEN_PROFILE.IND messages will be ignored!
                        repeat;
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
                        // CM_ATTEN_PROFILE.IND messages will be ignored!
                        repeat;
                     }
                     [] pt_SLAC_Port.receive {
                        setverdict(v_vct, "Invalid message type or content was received.");
                        v_repetition := false;
                     }
                     [] tc_TT_match_response.timeout {
                        log("TT_match_response timeout.");
                        if(v_count mod (par_C_EV_match_retry+1) == 0){
                            setverdict(v_vct,"The repetition limit is reached. " &
                                             "The Matching process is considered " &
                                             "as FAILED.");
                            v_repetition := false;
                        } else {
                          log("The repetition limit is not reached, " &
                              "a new CM_ATTEN_CHAR.IND message will be send.");
                        }
                    }
                }
            }
        }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_AttenuationCharacterization_002() runs on SLAC_Tester
                                                             return verdicttype {

       var MME v_responseMessage;
       var integer v_Num_soundsInt;
       var ResponseMessageList_TYPE reponseMessageList;
       var hexstring  v_variable;
       var integer v_count := 0;
       var boolean v_isRunning := true;
       var boolean v_repetition := true;
       var AttenProfile_TYPE v_attenuation_list;
       var integer v_count2 := 0;
       var integer v_countDecrement;
       var integer v_countStart;
       var integer v_countStop;
       var boolean v_firstSound := true;
```

```
tc_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min);

alt {
      []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_SLAC_PARM_REQ := '6064'H}),
                            md_CMN_CMN_CmSlacParmReq_001(
                            m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                            -> value v_responseMessage sender vc_sut_mac
                          {
        setverdict(pass,"CM_SLAC_PARM.REQ is correct.");
        vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
        repeat;
      }
      [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_START_ATTEN_CHAR_IND := '606A'H}),
                             md_CMN_CMN_CmStartAttenCharInd_001(
                             m_CMN_CMN_SlacPayloadHeader_001(), ?, ?,
                             '01'H, ?, vc_RunID)))
                             -> value v_responseMessage {

         if(v_count2 == 0) {
             tc_TT_EVSE_match_MNBC.start(par_TT_EVSE_match_MNBC);
             tc_TT_match_sequence.stop;

             vc_Num_sounds := v_responseMessage.mme_payload.payload.
                              cm_start_atten_char_ind.num_sounds;
             v_Num_soundsInt := hex2int(vc_Num_sounds);
         }
         setverdict(pass,"CM_START_ATTEN_CHAR.IND is correct.");
         v_count2 := v_count2 + 1;
         if(v_count2 < cc_numberOfStartAtten) {
             repeat;
         }
      }
      [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_SLAC_PARM_REQ := '6064'H}),
                             md_CMN_CMN_CmSlacParmReq_001(
                             m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

         tc_TT_match_sequence.start(par_TT_match_sequence);
         pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_SLAC_PARM_CNF := '6065'H}),
                          md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                          m_CMN_CMN_SlacPayloadHeader_001(),
                          vc_RunID)))
                          to vc_sut_mac;

         log("A further CM_SLAC_PARM.REQ message was received. " &
                "A new CM_SLAC_PARM.CNF has to be send.");
         repeat;
      }
      [] a_EVCC_processPLCLinkNotifications_002();
      [] pt_SLAC_Port.receive {
         setverdict(fail, "Invalid message type or content was received.");
         if(v_count2 < cc_numberOfStartAtten) {
             setverdict(fail, "A wrong number of CM_START_ATTEN_CHAR.IND " &
                              "message was received.");
         }
      }
      [] tc_TT_match_sequence.timeout {
          setverdict(fail,"TT_match_sequence timeout. " &
                           "No CM_START_ATTEN_CHAR.IND " &
                          "message was received. Matching process shall be " &
                          "considered as FAILED.");
          break;
      }
      [] tc_TT_EVSE_match_MNBC.timeout {
          setverdict(fail,"TT_EVSE_match_MNBC timeout. A wrong number of " &
                          "CM_START_ATTEN_CHAR.IND message was received. " &
                          "Matching process shall be considered as FAILED.");
      }
      [] tc_TT_EVSE_SLAC_init.timeout {
         setverdict(fail,"TT_EVSE_SLAC_init timeout. SECC assumes that " &
                         "no SLAC will be performed.");
        }
     }

    if(getverdict == pass) {
        while (v_isRunning) {
            alt {
                  [v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_MNBC_SOUND_IND := '6076'H}),
                                        md_CMN_CMN_CmMnbcSoundInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        ?, vc_RunID, ?)))
                                        -> value v_responseMessage {

                     v_countStart := hex2int(v_responseMessage.
                                            mme_payload.payload.
                                            cm_mnbc_sound_ind.count);
                     v_firstSound := false;
                     if(v_countStart == cc_numberOfSoundings) {
                         v_countDecrement := v_countStart - 1;
                         v_countStop := 1;
                     } else if(v_countStart == cc_numberOfSoundings - 1){
                         v_countDecrement := v_countStart - 1;
                         v_countStop := 0;
                     }
                     else {setverdict(fail, "The field 'count' has an " &
                                          "invalid value.");
                         v_isRunning := false;
                         break;
                     }
                     repeat;
                  }
                  [not v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_MNBC_SOUND_IND := '6076'H}),
                                       md_CMN_CMN_CmMnbcSoundInd_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(),
                                       int2hex(v_countDecrement,2),
                                       vc_RunID, ?))) {
                     v_countDecrement := v_countDecrement - 1;
                     repeat;
                  }
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_ATTEN_PROFILE_IND := '6086'H}),
                                        md_EVCC_CMN_CmAttenProfileInd_001(
                                        vc_sut_mac, ?, *)))
                                        -> value v_responseMessage {

                     if(ispresent(v_responseMessage.mme_payload.payload.
                                  cm_atten_profile_ind.attenuation_list)) {

                         if(v_responseMessage.mme_payload.payload.
                            cm_atten_profile_ind.num_groups != '3A'H) {
                            setverdict(fail,"Invalid numGroups value detected.");
                            v_isRunning := false;
                            break;
                         }
                         reponseMessageList[v_count] := v_responseMessage;
                         v_count := v_count + 1;
                     }
                     else {
                         log("Attenuation list was empty, the received message could not " &
                             "be considered for attenuation calculation.");
                     }
                  }
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_SLAC_PARM_REQ := '6064'H}),
                                        md_CMN_CMN_CmSlacParmReq_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                     setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                      "New Matching process is started.");
                     v_isRunning := false;
                  }
                  [] pt_SLAC_Port.receive {
                     setverdict(fail, "Invalid message type or content was received.");
```

```
                v_isRunning := false;
             }
             [] tc_TT_EVSE_match_MNBC.timeout {
                v_isRunning := false;
              }
           }
      if(v_count == v_Num_soundsInt){
          tc_TT_EVSE_match_MNBC.stop;
          v_isRunning := false;
      }
   }
}
if (v_count>0){
 if(v_countDecrement != (v_countStop - 1)) {
    setverdict(fail,"A wrong number of CM_MNBC_SOUND.IND messages " &
                    "was received.");
 } else {
    vc_Num_sounds := int2hex(v_count,2);
    setverdict(pass,"CM_MNBC_SOUND.IND is correct.");
    setverdict(pass,"CM_ATTEN_PROFILE.IND is correct.");
      v_attenuation_list := averageCalc(reponseMessageList, v_count);
 }
}
else {
    setverdict(fail,"No Atten Profile messages received.");
}

if(getverdict == pass) {
    v_count := 0;
    while(v_repetition){

        tc_TT_match_response.start(par_TT_match_response);
        v_count := v_count + 1;
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_ATTEN_CHAR_IND    := '606E'H}),
                         md_EVCC_CMN_CmAttenCharInd_001(
                         m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                         vc_RunID, vc_Num_sounds, v_attenuation_list)))
                         to vc_sut_mac;
         alt {
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_ATTEN_CHAR_RSP :='606F'H}),
                                       md_CMN_CMN_CmAttenCharRsp_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(),
                                       md_CMN_CMN_Acvarfield_001(
                                       vc_sut_mac, vc_RunID)))) {

                   setverdict(pass,"CM_ATTEN_CHAR.RSP is correct.");
                   v_repetition := false;
                   tc_TT_EVSE_match_session.start(par_TT_EVSE_match_session);
               }
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                         md_CMN_CMN_SlacMmeCmnHeader_001({
                                         CM_SLAC_PARM_REQ := '6064'H}),
                                       md_CMN_CMN_CmSlacParmReq_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                   setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                       "New Matching process is started.");
                   setverdict(fail, "SUT did not send a CM_ATTEN_CHAR.RSP " &
                                      "message to the second EVSE.");
                   v_repetition := false;
               }
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_MNBC_SOUND_IND := '6076'H}),?)) {
                  // CM_ATTEN_PROFILE.IND messages will be ignored!
                  repeat;
               }
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
                  // CM_ATTEN_PROFILE.IND messages will be ignored!
                  repeat;
               }
               [] pt_SLAC_Port.receive {
```

```
                        setverdict(fail, "Invalid message type or content was received.");
                        v_repetition := false;
                     }
                     [] tc_TT_match_response.timeout {
                        log("TT_match_response timeout.");
                        if(v_count mod (par_C_EV_match_retry+1) == 0){
                            setverdict(fail,"The repetition limit is reached. " &
                                            "The Matching process is considered " &
                                            "as FAILED.");
                            v_repetition := false;
                        } else {
                          log("The repetition limit is not reached, " &
                              "a new CM_ATTEN_CHAR.IND message will be send.");
                        }
                    }
               }
          }
      }
   return getverdict;
}

function f_EVCC_CMN_TB_VTB_AttenuationCharacterization_003() runs on SLAC_Tester
                                                             return verdicttype {

     var MME v_responseMessage;
     var integer v_Num_soundsInt;
     var ResponseMessageList_TYPE reponseMessageList;
     var hexstring  v_variable;
     var integer v_count := 0;
     var boolean v_isRunning := true;
     var boolean v_repetition := true;
     var AttenProfile_TYPE v_attenuation_list;
     timer t_TT_EVSE_SLAC_init;
     var integer v_count2 := 0;
     var integer v_countDecrement;
     var integer v_countStart;
     var integer v_countStop;
     var boolean v_firstSound := true;

     t_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min);

      alt{
            []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_SLAC_PARM_REQ := '6064'H}),
                                   md_CMN_CMN_CmSlacParmReq_001(
                                   m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                                   -> value v_responseMessage sender vc_sut_mac {

                setverdict(pass,"CM_SLAC_PARM.REQ is correct.");
                vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
                tc_TT_match_sequence.start(par_TT_match_sequence);
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_PARM_CNF := '6065'H}),
                                  md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                  to vc_sut_mac;
            }
            [] a_EVCC_processPLCLinkNotifications_002();
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] t_TT_EVSE_SLAC_init.timeout {
                setverdict(fail,"TT_EVSE_SLAC_init timeout. SECC assumes that no SLAC " &
                                "will be performed.");
            }
       }

       if(getverdict == pass) {

           alt {
                 [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_START_ATTEN_CHAR_IND := '606A'H}),
                                         md_CMN_CMN_CmStartAttenCharInd_001(
                                         m_CMN_CMN_SlacPayloadHeader_001(), ?, ?,
                                         '01'H, ?, vc_RunID)))
                                         -> value v_responseMessage {
```

```
            if(v_count2 == 0) {
                tc_TT_EVSE_match_MNBC.start(par_TT_EVSE_match_MNBC);
                tc_TT_match_sequence.stop;

                vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                 cm_start_atten_char_ind.num_sounds;
                v_Num_soundsInt := hex2int(vc_Num_sounds);
            }
            setverdict(pass,"CM_START_ATTEN_CHAR.IND is correct.");
            v_count2 := v_count2 + 1;
            if(v_count2 < cc_numberOfStartAtten) {
                repeat;
            }
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_PARM_REQ := '6064'H}),
                                 md_CMN_CMN_CmSlacParmReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))){

            tc_TT_match_sequence.start(par_TT_match_sequence);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_PARM_CNF := '6065'H}),
                               md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                               m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                               to vc_sut_mac;

            log("A further CM_SLAC_PARM.REQ message was received. " &
                 "A new CM_SLAC_PARM.CNF has to be send.");
            repeat;
         }
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
            if(v_count2 < cc_numberOfStartAtten) {
                setverdict(fail, "A wrong number of CM_START_ATTEN_CHAR.IND " &
                                 "message was received.");
            }
         }
         [] tc_TT_match_sequence.timeout {
            setverdict(fail,"TT_match_sequence timeout. " &
                            "No CM_START_ATTEN_CHAR.IND " &
                            "message was received. Matching process shall be " &
                            "considered as FAILED.");
            break;
         }
         [] tc_TT_EVSE_match_MNBC.timeout {
            setverdict(fail,"TT_EVSE_match_MNBC timeout. A wrong number of " &
                            "CM_START_ATTEN_CHAR.IND message was received. " &
                            "Matching process shall be considered as FAILED.");
         }
    }

   if(getverdict == pass) {
       while (v_isRunning) {
           alt {
                 [v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_MNBC_SOUND_IND := '6076'H}),
                                      md_CMN_CMN_CmMnbcSoundInd_001(
                                      m_CMN_CMN_SlacPayloadHeader_001(),
                                      ?, vc_RunID, ?)))
                                      -> value v_responseMessage {

                  v_countStart := hex2int(v_responseMessage.
                                          mme_payload.payload.
                                          cm_mnbc_sound_ind.count);
                   v_firstSound := false;
                   if(v_countStart == cc_numberOfSoundings) {
                       v_countDecrement := v_countStart - 1;
                       v_countStop := 1;
                   } else if(v_countStart == cc_numberOfSoundings - 1){
                       v_countDecrement := v_countStart - 1;
                       v_countStop := 0;
                   }
                   else {setverdict(fail, "The field 'count' has an " &
                                          "invalid value.");
```

```
                                v_isRunning := false;
                                break;
                            }
                            repeat;
                        }
                        [not v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_MNBC_SOUND_IND := '6076'H}),
                                           md_CMN_CMN_CmMnbcSoundInd_001(
                                           m_CMN_CMN_SlacPayloadHeader_001(),
                                           int2hex(v_countDecrement,2),
                                           vc_RunID, ?))) {
                         v_countDecrement := v_countDecrement - 1;
                         repeat;
                        }
                        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_ATTEN_PROFILE_IND := '6086'H}),
                                              md_EVCC_CMN_CmAttenProfileInd_001(
                                              vc_sut_mac, ?, *)))
                                              -> value v_responseMessage {

                           if(ispresent(v_responseMessage.mme_payload.payload.
                                        cm_atten_profile_ind.attenuation_list)) {

                               if(v_responseMessage.mme_payload.payload.
                                  cm_atten_profile_ind.num_groups != '3A'H) {
                                   setverdict(fail,"Invalid numGroups value detected.");
                                   v_isRunning := false;
                                   break;
                               }
                               reponseMessageList[v_count] := v_responseMessage;
                               v_count := v_count + 1;
                           }
                           else {
                               log("Attenuation list was empty, the received message " &
                                   "could not be considered for attenuation calculation.");
                           }
                        }
                        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_SLAC_PARM_REQ := '6064'H}),
                                              md_CMN_CMN_CmSlacParmReq_001(
                                              m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                           setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                             "New Matching process is started.");
                           v_isRunning := false;
                        }
                        [] pt_SLAC_Port.receive {
                           setverdict(fail, "Invalid message type or content was received.");
                           v_isRunning := false;
                        }
                        [] tc_TT_EVSE_match_MNBC.timeout {
                           v_isRunning := false;
                         }
                      }
                  if(v_count == v_Num_soundsInt){
                      tc_TT_EVSE_match_MNBC.stop;
                      v_isRunning := false;
                  }
                }
            }
            if (v_count>0){
             if(v_countDecrement != (v_countStop - 1)) {
                setverdict(fail,"A wrong number of CM_MNBC_SOUND.IND messages " &
                                "was received.");
             } else {
                vc_Num_sounds := int2hex(v_count,2);
                setverdict(pass,"CM_MNBC_SOUND.IND is correct.");
                setverdict(pass,"CM_ATTEN_PROFILE.IND is correct.");
                  v_attenuation_list := averageCalc(reponseMessageList, v_count);
             }
            }
            else {
                setverdict(fail,"No Atten Profile messages received.");
            }
        }
        return getverdict;
```

```
}

function f_EVCC_CMN_TB_VTB_AttenuationCharacterization_004() runs on EVCC_Tester
                                                             return verdicttype {

       var MME v_responseMessage;
       var integer v_Num_soundsInt;
       var ResponseMessageList_TYPE reponseMessageList;
       var hexstring  v_variable;
       var integer v_count := 0;
       var boolean v_isRunning := true;
       var AttenProfile_TYPE v_attenuation_list;
       var integer v_count2 := 0;
       var integer v_countDecrement;
       var integer v_countStart;
       var integer v_countStop;
       var boolean v_firstSound := true;

       tc_TT_match_sequence.start(par_TT_match_sequence);
       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_SLAC_PARM_CNF := '6065'H}),
                         md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                         m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                         to vc_sut_mac;

       alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_START_ATTEN_CHAR_IND := '606A'H}),
                                     md_CMN_CMN_CmStartAttenCharInd_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), ?, ?,
                                     '01'H, ?, vc_RunID)))
                                     -> value v_responseMessage {

                if(v_count2 == 0) {
                    tc_TT_EVSE_match_MNBC.start(par_TT_EVSE_match_MNBC);
                    tc_TT_match_sequence.stop;
                    tc_TP_EVSE_atten_results.start(par_TT_EV_atten_results);
                    vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                     cm_start_atten_char_ind.num_sounds;
                    v_Num_soundsInt := hex2int(vc_Num_sounds);
                }
                setverdict(pass,"CM_START_ATTEN_CHAR.IND is correct.");
                v_count2 := v_count2 + 1;
                if(v_count2 < cc_numberOfStartAtten) {
                    repeat;
                }
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_SLAC_PARM_REQ := '6064'H}),
                                     md_CMN_CMN_CmSlacParmReq_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                tc_TT_match_sequence.start(par_TT_match_sequence);
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_PARM_CNF := '6065'H}),
                                  md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                                  to vc_sut_mac;

                log("A further CM_SLAC_PARM.REQ message was received. " &
                    "A new CM_SLAC_PARM.CNF has to be send.");
                repeat;
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
                if(v_count2 < cc_numberOfStartAtten) {
                    setverdict(fail, "A wrong number of CM_START_ATTEN_CHAR.IND " &
                                     "message was received.");
                }
             }
             [] tc_TT_match_sequence.timeout {
                setverdict(fail,"TT_match_sequence timeout. " &
                                "No CM_START_ATTEN_CHAR.IND " &
                                "message was received. Matching process shall be " &
```

```
                        "considered as FAILED.");
                break;
             }
             [] tc_TT_EVSE_match_MNBC.timeout {
                setverdict(fail,"TT_EVSE_match_MNBC timeout. A wrong number of " &
                                "CM_START_ATTEN_CHAR.IND message was received. " &
                                "Matching process shall be considered as FAILED.");
             }
         }
        if(getverdict == pass) {
            while (v_isRunning) {
                alt {
                      [v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                              md_CMN_CMN_SlacMmeCmnHeader_001({
                                              CM_MNBC_SOUND_IND := '6076'H}),
                                              md_CMN_CMN_CmMnbcSoundInd_001(
                                              m_CMN_CMN_SlacPayloadHeader_001(),
                                              ?, vc_RunID, ?)))
                                              -> value v_responseMessage {

                         v_countStart := hex2int(v_responseMessage.
                                                   mme_payload.payload.
                                                   cm_mnbc_sound_ind.count);
                         v_firstSound := false;
                         if(v_countStart == cc_numberOfSoundings) {
                             v_countDecrement := v_countStart - 1;
                             v_countStop := 1;
                         } else if(v_countStart == cc_numberOfSoundings - 1){
                             v_countDecrement := v_countStart - 1;
                             v_countStop := 0;
                         }
                         else {setverdict(fail, "The field 'count' has an " &
                                                "invalid value.");
                             v_isRunning := false;
                             break;
                         }
                         repeat;
                      }
                      [not v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                             md_CMN_CMN_SlacMmeCmnHeader_001({
                                             CM_MNBC_SOUND_IND := '6076'H}),
                                             md_CMN_CMN_CmMnbcSoundInd_001(
                                             m_CMN_CMN_SlacPayloadHeader_001(),
                                             int2hex(v_countDecrement,2),
                                             vc_RunID, ?))) {
                         v_countDecrement := v_countDecrement - 1;
                         repeat;
                      }
                      [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                                 CM_ATTEN_PROFILE_IND := '6086'H}),
                                                md_EVCC_CMN_CmAttenProfileInd_001(
                                                vc_sut_mac, ?, *)))
                                                -> value v_responseMessage {

                         if(ispresent(v_responseMessage.mme_payload.payload.
                                      cm_atten_profile_ind.attenuation_list)) {

                             if(v_responseMessage.mme_payload.payload.
                                cm_atten_profile_ind.num_groups != '3A'H) {
                                 setverdict(fail,"Invalid numGroups value detected.");
                                 v_isRunning := false;
                                 break;
                             }
                             reponseMessageList[v_count] := v_responseMessage;
                             v_count := v_count + 1;
                         }
                         else {
                             log("Attenuation list was empty, the received message could not " &
                                 "be considered for attenuation calculation.");
                         }
                      }
                      [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                                 CM_SLAC_PARM_REQ := '6064'H}),
                                                md_CMN_CMN_CmSlacParmReq_001(
                                                m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                         setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                  "New Matching process is started.");
                        v_isRunning := false;
                     }
                     [] pt_SLAC_Port.receive {
                        setverdict(fail, "Invalid message type or content was received.");
                        v_isRunning := false;
                     }
                     [] tc_TT_EVSE_match_MNBC.timeout {
                        v_isRunning := false;
                      }
                     }
                 if(v_count == v_Num_soundsInt){
                     tc_TT_EVSE_match_MNBC.stop;
                     v_isRunning := false;
                 }
              }
         }
         if (v_count>0){
          if(v_countDecrement != (v_countStop - 1)) {
             setverdict(fail,"A wrong number of CM_MNBC_SOUND.IND messages " &
                             "was received.");
          } else {
             vc_Num_sounds := int2hex(v_count,2);
             setverdict(pass,"CM_MNBC_SOUND.IND is correct.");
             setverdict(pass,"CM_ATTEN_PROFILE.IND is correct.");
               v_attenuation_list := averageCalc(reponseMessageList, v_count);
          }
         }
         else {
             setverdict(fail,"No Atten Profile messages received.");
         }
         if(getverdict == pass) {
              // wait until tc_TP_EVSE_atten_results timer expires
              alt {
                  [] tc_TP_EVSE_atten_results.timeout;
                  [] pt_SLAC_Port.receive {
                      setverdict(fail, "Invalid message type or content " &
                                       "was received.");
                  }
              }

              tc_TT_match_response.start(par_TT_match_response);
              pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_IND := '606E'H}),
                                  md_EVCC_CMN_CmAttenCharInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                                  vc_RunID, vc_Num_sounds, v_attenuation_list)))
                                  to vc_sut_mac;
               alt {
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_ATTEN_CHAR_RSP :='606F'H}),
                                               md_CMN_CMN_CmAttenCharRsp_001(
                                               m_CMN_CMN_SlacPayloadHeader_001(),
                                               md_CMN_CMN_Acvarfield_001(
                                               vc_sut_mac, vc_RunID)))) {

                         setverdict(fail,"CM_ATTEN_CHAR.RSP message was not expected.");
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_SLAC_PARM_REQ := '6064'H}),
                                               md_CMN_CMN_CmSlacParmReq_001(
                                               m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                         setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                            "New Matching process is started.");
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_MNBC_SOUND_IND := '6076'H}),?)) {
                        // CM_ATTEN_PROFILE.IND messages will be ignored!
                        repeat;
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                               md_CMN_CMN_SlacMmeCmnHeader_001({
                                               CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
```

```
                    // CM_ATTEN_PROFILE.IND messages will be ignored!
                    repeat;
                }
                [] pt_SLAC_Port.receive {
                     setverdict(fail, "Invalid message type or content was received.");
                }
                [] tc_TT_match_response.timeout {
                     setverdict(pass, "TT_match_response timeout. " &
                                         "No CM_ATTEN_CHAR.RSP message was " &
                                      "received from the SUT.");
                }
            }
        }
        return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_AttenuationCharacterization_005(in template MME_Payload v_payload)
                                                               runs on EVCC_Tester
                                                               return verdicttype{

        var MME v_responseMessage;
        var integer v_Num_soundsInt;
        var ResponseMessageList_TYPE reponseMessageList;
        var hexstring  v_variable;
        var integer v_count := 0;
        var boolean v_isRunning := true;
        var AttenProfile_TYPE v_attenuation_list;
        var integer v_count2 := 0;
        var integer v_countDecrement;
        var integer v_countStart;
        var integer v_countStop;
        var boolean v_firstSound := true;

        tc_TT_match_sequence.start(par_TT_match_sequence);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_SLAC_PARM_CNF := '6065'H}),
                       md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                       m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                       to vc_sut_mac;

        alt {
              [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_START_ATTEN_CHAR_IND := '606A'H}),
                                   md_CMN_CMN_CmStartAttenCharInd_001(
                                   m_CMN_CMN_SlacPayloadHeader_001(), ?, ?,
                                   '01'H, ?, vc_RunID)))
                                   -> value v_responseMessage {

                 if(v_count2 == 0) {
                     tc_TT_EVSE_match_MNBC.start(par_TT_EVSE_match_MNBC);
                     tc_TT_match_sequence.stop;
                     tc_TP_EVSE_atten_results.start(par_TT_EV_atten_results);
                     vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                      cm_start_atten_char_ind.num_sounds;
                     if(not(ispresent(v_payload.payload.cm_atten_char_ind.num_sounds))) {
                         v_payload.payload.cm_atten_char_ind.num_sounds := vc_Num_sounds;
                     }
                     v_Num_soundsInt := hex2int(vc_Num_sounds);
                 }
                 setverdict(pass,"CM_START_ATTEN_CHAR.IND is correct.");
                 v_count2 := v_count2 + 1;
                 if(v_count2 < cc_numberOfStartAtten) {
                     repeat;
                 }
              }
              [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_SLAC_PARM_REQ := '6064'H}),
                                   md_CMN_CMN_CmSlacParmReq_001(
                                   m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                 tc_TT_match_sequence.start(par_TT_match_sequence);
                 pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                   md_CMN_CMN_SlacMmeCmnHeader_001({
                                   CM_SLAC_PARM_CNF := '6065'H}),
                                md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                                m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                         to vc_sut_mac;

              log("A further CM_SLAC_PARM.REQ message was received. " &
                     "A new CM_SLAC_PARM.CNF has to be send.");
              repeat;
           }
           [] pt_SLAC_Port.receive {
              setverdict(fail, "Invalid message type or content was received.");
              if(v_count2 < cc_numberOfStartAtten) {
                  setverdict(fail, "A wrong number of CM_START_ATTEN_CHAR.IND " &
                                   "message was received.");
              }
           }
           [] tc_TT_match_sequence.timeout {
              setverdict(fail,"TT_match_sequence timeout. " &
                              "No CM_START_ATTEN_CHAR.IND " &
                              "message was received. Matching process shall be " &
                              "considered as FAILED.");
              break;
           }
           [] tc_TT_EVSE_match_MNBC.timeout {
              setverdict(fail,"TT_EVSE_match_MNBC timeout. A wrong number of " &
                              "CM_START_ATTEN_CHAR.IND message was received. " &
                              "Matching process shall be considered as FAILED.");
           }
     }
     if(getverdict == pass) {
        while (v_isRunning) {
            alt {
                  [v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_MNBC_SOUND_IND := '6076'H}),
                                       md_CMN_CMN_CmMnbcSoundInd_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(),
                                       ?, vc_RunID, ?)))
                                       -> value v_responseMessage {

                     v_countStart := hex2int(v_responseMessage.
                                             mme_payload.payload.
                                             cm_mnbc_sound_ind.count);
                     v_firstSound := false;
                     if(v_countStart == cc_numberOfSoundings) {
                         v_countDecrement := v_countStart - 1;
                         v_countStop := 1;
                     } else if(v_countStart == cc_numberOfSoundings - 1){
                         v_countDecrement := v_countStart - 1;
                         v_countStop := 0;
                     }
                     else {setverdict(fail, "The field 'count' has an " &
                                            "invalid value.");
                         v_isRunning := false;
                         break;
                     }
                     repeat;
                  }
                  [not v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_MNBC_SOUND_IND := '6076'H}),
                                      md_CMN_CMN_CmMnbcSoundInd_001(
                                      m_CMN_CMN_SlacPayloadHeader_001(),
                                      int2hex(v_countDecrement,2),
                                      vc_RunID, ?))) {
                     v_countDecrement := v_countDecrement - 1;
                     repeat;
                  }
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_ATTEN_PROFILE_IND := '6086'H}),
                                         md_EVCC_CMN_CmAttenProfileInd_001(
                                         vc_sut_mac, ?, *)))
                                         -> value v_responseMessage {

                     if(ispresent(v_responseMessage.mme_payload.payload.
                                  cm_atten_profile_ind.attenuation_list)) {

                         if(v_responseMessage.mme_payload.payload.
                            cm_atten_profile_ind.num_groups != '3A'H) {
                             setverdict(fail,"Invalid numGroups value detected.");
```

```
                        v_isRunning := false;
                        break;
                    }
                    reponseMessageList[v_count] := v_responseMessage;
                    v_count := v_count + 1;
                }
                else {
                    log("Attenuation list was empty, the received message could not " &
                        "be considered for attenuation calculation.");
                }
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_PARM_REQ := '6064'H}),
                                 md_CMN_CMN_CmSlacParmReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                 "New Matching process is started.");
                v_isRunning := false;
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
                v_isRunning := false;
             }
             [] tc_TT_EVSE_match_MNBC.timeout {
                v_isRunning := false;
              }
          }
       if(v_count == v_Num_soundsInt){
           tc_TT_EVSE_match_MNBC.stop;
           v_isRunning := false;
       }
     }
  }
  if (v_count>0){
   if(v_countDecrement != (v_countStop - 1)) {
      setverdict(fail,"A wrong number of CM_MNBC_SOUND.IND messages " &
                      "was received.");
   } else {
      vc_Num_sounds := int2hex(v_count,2);
      setverdict(pass,"CM_MNBC_SOUND.IND is correct.");
      setverdict(pass,"CM_ATTEN_PROFILE.IND is correct.");
        v_attenuation_list := averageCalc(reponseMessageList, v_count);
   }
  }
  else {
      setverdict(fail,"No Atten Profile messages received.");
  }
  if(getverdict == pass) {
      // send invalid CM_ATTEN_CHAR.IND message
      pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_ATTEN_CHAR_IND := '606E'H}), v_payload))
                          to vc_sut_mac;
      alt {

         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_RSP :='606F'H}),?)) {

             setverdict(fail,"Invalid CM_ATTEN_CHAR.IND messages shall be ignored.");
         }
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
         }
         [] tc_TP_EVSE_atten_results.timeout {

            tc_TT_match_response.start(par_TT_match_response);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                               md_CMN_CMN_SlacMmeCmnHeader_001({
                               CM_ATTEN_CHAR_IND := '606E'H}),
                               md_EVCC_CMN_CmAttenCharInd_001(
                               m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                               vc_RunID, vc_Num_sounds, v_attenuation_list)))
                               to vc_sut_mac;
            alt {
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
```

```
                                          CM_ATTEN_CHAR_RSP :='606F'H}),
                                          md_CMN_CMN_CmAttenCharRsp_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(),
                                          md_CMN_CMN_Acvarfield_001(
                                          vc_sut_mac, vc_RunID)))) {

                         setverdict(fail,"CM_ATTEN_CHAR.RSP message was not expected.");
                        }
                        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_SLAC_PARM_REQ := '6064'H}),
                                          md_CMN_CMN_CmSlacParmReq_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                           setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                            "New Matching process is started.");
                        }
                        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_MNBC_SOUND_IND := '6076'H}),?)) {
                           // CM_ATTEN_PROFILE.IND messages will be ignored!
                           repeat;
                        }
                        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
                           // CM_ATTEN_PROFILE.IND messages will be ignored!
                           repeat;
                        }
                        [] pt_SLAC_Port.receive {
                           setverdict(fail, "Invalid message type or content was received.");
                        }
                        [] tc_TT_match_response.timeout {
                            setverdict(pass, "TT_match_response timeout. " &
                                              "No CM_ATTEN_CHAR.RSP message was " &
                                           "received from the SUT.");
                        }
                  }
             }
         }
     }
     return getverdict;
 }

 function f_EVCC_CMN_TB_VTB_AttenuationCharacterization_006(HAL_61851_PwmMode_Type pwmMode)
                                                            runs on EVCC_Tester
                                                            return verdicttype {

        // set error state
            f_EVCC_changeValidStateCondition(E,E);
            f_EVCC_setPwmMode(pwmMode);

        tc_TT_match_sequence.start(par_TT_match_sequence);
        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_SLAC_PARM_CNF := '6065'H}),
                          md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                          m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                          to vc_sut_mac;

        alt {
              [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_START_ATTEN_CHAR_IND := '606A'H}),?)) {

                 setverdict(fail,"CM_START_ATTEN_CHAR.IND message " &
                                 "was not expected. CP State E/F " &
                                 "should be detected before.");
                 tc_TT_match_sequence.stop;
              }
              [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SLAC_PARM_REQ := '6064'H}),
                                      md_CMN_CMN_CmSlacParmReq_001(
                                      m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                 tc_TT_match_sequence.start(par_TT_match_sequence);
                 pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_SLAC_PARM_CNF := '6065'H}),
                             md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                             m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                             to vc_sut_mac;

                  log("A further CM_SLAC_PARM.REQ message was received. " &
                         "A new CM_SLAC_PARM.CNF has to be send.");
                  repeat;
               }
               [] pt_SLAC_Port.receive {
                  setverdict(fail, "Invalid message type or content was received.");
               }
               [] tc_TT_match_sequence.timeout {
                  setverdict(pass,"TT_match_sequence timer has expired, " &
                                  "the Matching process was terminated " &
                                  "by the SUT.");
               }
      }
      return getverdict;
   }

   function f_EVCC_AC_TB_VTB_AttenuationCharacterization_001(in integer v_dutcCycle)
                                                             runs on EVCC_Tester
                                                             return verdicttype {

        var MME v_responseMessage;
          var integer v_Num_soundsInt;
          var ResponseMessageList_TYPE reponseMessageList;
          var hexstring  v_variable;
          var integer v_count := 0;
          var boolean v_isRunning := true;
          var boolean v_repetition := true;
          var AttenProfile_TYPE v_attenuation_list;
          var integer v_count2 := 0;
          var integer v_countDecrement;
          var integer v_countStart;
          var integer v_countStop;
          var boolean v_firstSound := true;

             f_EVCC_setDutyCycle(v_dutcCycle);

          tc_TT_match_sequence.start(par_TT_match_sequence);
          pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                            md_CMN_CMN_SlacMmeCmnHeader_001({
                            CM_SLAC_PARM_CNF := '6065'H}),
                            md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                            m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                            to vc_sut_mac;

          alt {
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_START_ATTEN_CHAR_IND := '606A'H}),
                                md_CMN_CMN_CmStartAttenCharInd_001(
                                m_CMN_CMN_SlacPayloadHeader_001(), ?, ?,
                                '01'H, ?, vc_RunID)))
                                -> value v_responseMessage {

                  if(v_count2 == 0) {
                      tc_TT_EVSE_match_MNBC.start(par_TT_EVSE_match_MNBC);
                      tc_TT_match_sequence.stop;

                      vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                       cm_start_atten_char_ind.num_sounds;
                      v_Num_soundsInt := hex2int(vc_Num_sounds);
                  }
                  setverdict(pass,"CM_START_ATTEN_CHAR.IND is correct.");
                  v_count2 := v_count2 + 1;
                  if(v_count2 < cc_numberOfStartAtten) {
                      repeat;
                  }
               }
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_SLAC_PARM_REQ := '6064'H}),
                                md_CMN_CMN_CmSlacParmReq_001(
                                m_CMN_CMN_SlacPayloadHeader_001(), ?))){
```

```
            tc_TT_match_sequence.start(par_TT_match_sequence);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_SLAC_PARM_CNF := '6065'H}),
                              md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                              m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                              to vc_sut_mac;

            log("A further CM_SLAC_PARM.REQ message was received. " &
                "A new CM_SLAC_PARM.CNF has to be send.");
            repeat;
         }
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
            if(v_count2 < cc_numberOfStartAtten) {
                setverdict(fail, "A wrong number of CM_START_ATTEN_CHAR.IND " &
                                 "message was received.");
            }
         }
         [] tc_TT_match_sequence.timeout {
            setverdict(fail,"TT_match_sequence timeout. " &
                            "No CM_START_ATTEN_CHAR.IND " &
                            "message was received. Matching process shall be " &
                            "considered as FAILED.");
            break;
         }
         [] tc_TT_EVSE_match_MNBC.timeout {
            setverdict(fail,"TT_EVSE_match_MNBC timeout. A wrong number of " &
                            "CM_START_ATTEN_CHAR.IND message was received. " &
                            "Matching process shall be considered as FAILED.");
         }
     }

    if(getverdict == pass) {
        while (v_isRunning) {
            alt {
                  [v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_MNBC_SOUND_IND := '6076'H}),
                                        md_CMN_CMN_CmMnbcSoundInd_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(),
                                        ?, vc_RunID, ?)))
                                        -> value v_responseMessage {

                     v_countStart := hex2int(v_responseMessage.
                                             mme_payload.payload.
                                             cm_mnbc_sound_ind.count);
                     v_firstSound := false;
                     if(v_countStart == cc_numberOfSoundings) {
                         v_countDecrement := v_countStart - 1;
                         v_countStop := 1;
                     } else if(v_countStart == cc_numberOfSoundings - 1){
                         v_countDecrement := v_countStart - 1;
                         v_countStop := 0;
                     }
                     else {setverdict(fail, "The field 'count' has an " &
                                            "invalid value.");
                         v_isRunning := false;
                         break;
                     }
                     repeat;
                  }
                  [not v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_MNBC_SOUND_IND := '6076'H}),
                                       md_CMN_CMN_CmMnbcSoundInd_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(),
                                       int2hex(v_countDecrement,2),
                                       vc_RunID, ?))) {
                     v_countDecrement := v_countDecrement - 1;
                     repeat;
                  }
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_ATTEN_PROFILE_IND := '6086'H}),
                                          md_EVCC_CMN_CmAttenProfileInd_001(
                                          vc_sut_mac, ?, *)))
                                          -> value v_responseMessage {
```

```
                  if(ispresent(v_responseMessage.mme_payload.payload.
                               cm_atten_profile_ind.attenuation_list)) {

                      if(v_responseMessage.mme_payload.payload.
                         cm_atten_profile_ind.num_groups != '3A'H) {
                          setverdict(fail,"Invalid numGroups value detected.");
                          v_isRunning := false;
                          break;
                      }
                      reponseMessageList[v_count] := v_responseMessage;
                      v_count := v_count + 1;
                  }
                  else {
                      log("Attenuation list was empty, the received message could not " &
                          "be considered for attenuation calculation.");
                  }
               }
               [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SLAC_PARM_REQ := '6064'H}),
                                       md_CMN_CMN_CmSlacParmReq_001(
                                       m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                  setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                    "New Matching process is started.");
                  v_isRunning := false;
               }
               [] pt_SLAC_Port.receive {
                  setverdict(fail, "Invalid message type or content was received.");
                  v_isRunning := false;
               }
               [] tc_TT_EVSE_match_MNBC.timeout {
                  v_isRunning := false;
                }
             }
         if(v_count == v_Num_soundsInt){
             tc_TT_EVSE_match_MNBC.stop;
             v_isRunning := false;
         }
      }
   }
   if (v_count>0){
    if(v_countDecrement != (v_countStop - 1)) {
       setverdict(fail,"A wrong number of CM_MNBC_SOUND.IND messages " &
                       "was received.");
    } else {
       vc_Num_sounds := int2hex(v_count,2);
       setverdict(pass,"CM_MNBC_SOUND.IND is correct.");
       setverdict(pass,"CM_ATTEN_PROFILE.IND is correct.");
         v_attenuation_list := averageCalc(reponseMessageList, v_count);

    }
   }
   else {
       setverdict(fail,"No Atten Profile messages received.");
   }

   if(getverdict == pass) {
       v_count := 0;
       while(v_repetition){

           tc_TT_match_response.start(par_TT_match_response);
           v_count := v_count + 1;
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_ATTEN_CHAR_IND := '606E'H}),
                             md_EVCC_CMN_CmAttenCharInd_001(
                             m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                             vc_RunID, vc_Num_sounds, v_attenuation_list)))
                             to vc_sut_mac;
            alt {
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_ATTEN_CHAR_RSP :='606F'H}),
                                          md_CMN_CMN_CmAttenCharRsp_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(),
                                          md_CMN_CMN_Acvarfield_001(
                                          vc_sut_mac, vc_RunID)))) {
```

```
                             setverdict(pass,"CM_ATTEN_CHAR.RSP is correct. " &
                                             "The change of the duty cycle " &
                                             "should not influence the EVCC Matching process.");
                             v_repetition := false;
                             tc_TT_EVSE_match_session.start(par_TT_EVSE_match_session);
                         }
                         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                                 CM_SLAC_PARM_REQ := '6064'H}),
                                                 md_CMN_CMN_CmSlacParmReq_001(
                                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

                             setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                                 "New Matching process is started.");
                             v_repetition := false;
                         }
                         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                                 CM_MNBC_SOUND_IND := '6076'H}),?)) {
                            // CM_ATTEN_PROFILE.IND messages will be ignored!
                            repeat;
                         }
                         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                                 CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
                            // CM_ATTEN_PROFILE.IND messages will be ignored!
                            repeat;
                         }
                         [] pt_SLAC_Port.receive {
                            setverdict(fail, "Invalid message type or content was received.");
                            v_repetition := false;
                         }
                         [] tc_TT_match_response.timeout {
                            log("TT_match_response timeout.");
                            if(v_count mod (par_C_EV_match_retry+1) == 0){
                                setverdict(fail,"The repetition limit is reached. " &
                                                "The Matching process is considered " &
                                                "as FAILED.");
                                v_repetition := false;
                            } else {
                              log("The repetition limit is not reached, " &
                                  "a new CM_ATTEN_CHAR.IND message will be send.");
                            }
                        }
                   }
              }
          }
       return getverdict;
   }
}
```

E.2.3 EVCC functions for CmValidate

```
module TestBehavior_EVCC_CmValidate {

    import from Timer_15118_3 all;
    import from Pixit_15118_3 all;
    import from Pics_15118_3 all;
    import from Templates_CMN_CmValidate all;
    import from Templates_CMN_CmStartAttenCharInd all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmMnbcSoundInd all;
    import from Templates_CMN_CmAttenCharRsp all;
    import from Templates_EVCC_CmAttenCharInd all;
    import from Templates_EVCC_CmAttenProfileInd all;
    import from Templates_CMN_CmSlacParm all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from TestBehavior_EVCC_AttenuationCharacterization all;
    import from LibFunctions_15118_3  { group generalFunctions; }
    import from Services_HAL_61851 all;
    import from Templates_CMN_CmSlacParm all;
    import from Templates_CMN_HAL61851 all;
    import from DataStructure_HAL_61851 all;
    import from Templates_CMN_CmSlacMatch all;
    import from Services_PLCLinkStatus all;
    import from Pics_15118 all;
    import from DataStructure_HAL_61851 all;
    import from Timer_15118 all;

    // EVCC Tester
    function f_EVCC_CMN_TB_VTB_CmValidate_001(in HAL_61851_Listener v_HAL_61851_Listener,
                                              in boolean v_changeDC, in integer v_dutyCycle,
                                              in verdicttype v_vct)
                                              runs on EVCC_Tester return verdicttype {

        var MME v_requestMessage;
        var integer cnt := 0;
        var boolean isStep2 := false;
        var float v_TT_EVSE_vald_toggle;

        alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_REQ := '6078'H}),
                                     md_CMN_CMN_CmValidateReq_004(?,?,?)))
                                     -> value v_requestMessage {

                var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                    cm_validate_req.vrVarField.pilot_timer;
                var SignalType_TYPE p_signalType := v_requestMessage.mme_payload.payload.
                                                    cm_validate_req.signalType;

                if((p_signalType != '00'H)) {
                    setverdict(v_vct,"Step 1 CM_VALIDATE.REQ is not correct. " &
                                     "Invalid signalType was detected.");
                }
                tc_TT_EVSE_match_session.stop;
                if(v_pilotTimer == '00'H) {
                    if(not isStep2) {
                        setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
                        isStep2 := true;
                        if(v_changeDC) {
                            // change current duty cycle
                            f_EVCC_setDutyCycle(v_dutyCycle);
                        }
                    }
                    else {
                        log("Step 2 CM_VALIDATE.REQ message contains timer field equal to zero. " &
                            "Step 1 CM_VALIDATE.CNF will be resent.");
                    }

                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_CNF := '6079'H}),
                                      md_CMN_CMN_CmValidateCnf_001(
                                      par_cmValidate_result_ready)))
                                      to vc_sut_mac;
                tc_TT_match_sequence.start(par_TT_match_sequence);
                repeat;
            }
            else if(decodeValdToggleTime(v_pilotTimer) >= par_TP_EV_vald_toggle_min and
                    decodeValdToggleTime(v_pilotTimer) <= par_TP_EV_vald_toggle_max and
                    isStep2) {

                setverdict(pass,"Step 2 CM_VALIDATE.REQ is correct.");
                tc_TT_match_sequence.stop;
                v_TT_EVSE_vald_toggle := decodeValdToggleTime(v_pilotTimer);
            }
            else {
                setverdict(v_vct, "Invalid message content was " &
                                  "received from the SUT.");
            }
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_REQ := '6078'H}),
                                 mw_CMN_CMN_CmValidateReq_003())) {

                setverdict(v_vct,"Result field is not set to 'Ready'. " &
                                 "Matching process is " &
                                 "considered as FAILED.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_PARM_REQ := '6064'H}),
                                 md_CMN_CMN_CmSlacParmReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))){

            setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                              "New Matching process is started.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_RSP := '606F'H}),?)) {
            // CM_ATTEN_CHAR.RSP messages will be ignored!
            repeat;
         }
         [] pt_SLAC_Port.receive {
            setverdict(v_vct, "Invalid message type or content was received.");
         }
         [] tc_TT_match_sequence.timeout {
            cnt := cnt +1;
            log("TT_match_sequence timeout.");
            if (cnt > par_C_EV_match_retry) {
                setverdict(v_vct,"Repetition limit is reached. " &
                                 "Matching process is " &
                                 "considered as FAILED.");
            }
            else {
                log("A new CM_VALIDATE.CNF message will be sent.");
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(
                                  par_cmValidate_result_ready)))
                                  to vc_sut_mac;

                tc_TT_match_sequence.start(par_TT_match_sequence);
                repeat;
            }
         }
         [] tc_TT_EVSE_match_session.timeout {
            setverdict(v_vct,"TT_EVSE_match_session timeout. " &
                             "Matching process shall " &
                             "be considered as FAILED.");
         }
    }

    if(getverdict == pass) {
        // BCB toggle sequence detection
        tc_TT_EVSE_vald_toggle.start(v_TT_EVSE_vald_toggle);
        f_EVCC_changeValidStateCondition(B,C);
        timer statetimer := (par_T_vald_state_duration_max + par_CMN_Transmission_Delay);
        statetimer.start;
```

```
            var integer toggleCnt := 0;

            alt {
               [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, B){

                   statetimer.stop;
                   toggleCnt := toggleCnt + 1;
                   f_EVCC_changeValidStateCondition(B,C);
                   repeat;
               }
               [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, C){

                   statetimer.stop;
                   f_EVCC_changeValidStateCondition(C,B);
                   statetimer.start(par_T_vald_state_duration_max);
                   repeat;
               }
               [] pt_HAL_61851_Internal_Port.receive {
                   setverdict(v_vct, "Received state has an invalid value.");
               }
               [] statetimer.timeout {
                   setverdict(v_vct, "The EVSE could not detect the corresponding " &
                                     "toggle value within " &
                                     "the maximal valid state duration.");
               }
               [] tc_TT_EVSE_vald_toggle.timeout {

                   if(toggleCnt > 0 and toggleCnt <= 3) {

                       setverdict(pass,"Valid BCB toggle sequence could be detected.");
                       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                             md_CMN_CMN_SlacMmeCmnHeader_001({
                                             CM_VALIDATE_CNF := '6079'H}),
                                          md_CMN_CMN_CmValidateCnf_002(
                                          int2hex(toggleCnt,2),
                                          par_cmValidate_result_success)))
                                          to vc_sut_mac;

                       f_EVCC_changeValidStateCondition(C,B);
                       tc_TT_match_sequence.start(par_TT_match_sequence);
                   }
                   else {
                      setverdict(v_vct,"Invalid BCB toggle sequence was detected.");
                   }
               }
            }
        }
        return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmValidate_002() runs on EVCC_Tester
                                                return verdicttype {

        var MME v_responseMessage;
        var integer v_count := 0;

        alt{
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_VALIDATE_REQ := '6078'H}),
                                    m_CMN_CMN_CmValidateReq_001()))
                                    -> value v_responseMessage {

                if(v_count > 0){
                    setverdict(pass,"CM_VALIDATE.REQ message was repeated.",v_count);
                }
                else {
                    tc_TT_EVSE_match_session.stop;
                }
                v_count := v_count + 1;
                tc_TT_match_response.start(par_TT_match_response + par_CMN_Transmission_Delay);

                if(v_count > par_C_EV_match_retry) {
                  alt{
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_VALIDATE_REQ := '6078'H}),
                                            m_CMN_CMN_CmValidateReq_001())) {
                      setverdict(fail,"CM_VALIDATE.REQ message was repeated, but v_count > " &
                              "par_C_EV_match_retry.");
              }
              [] tc_TT_match_response.timeout {
                setverdict(pass,"TT_match_response timeout. " &
                                 "The total number of retries is reached, " &
                                 "the Validation process " &
                                 "shall be considered as FAILED.");
               }
            }
          }
          else{
              repeat;
          }
       }
       [] pt_SLAC_Port.receive {
          setverdict(fail, "Invalid message type or content was received.");
       }
       [] tc_TT_EVSE_match_session.timeout {
          setverdict(fail,"TT_EVSE_match_session timeout. Matching process shall be " &
                          "considered as FAILED.");
       }
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmValidate_003(in template(present) MME_Payload v_validateCnf)
                                          runs on EVCC_Tester return verdicttype {

    var MME v_responseMessage;
    var integer v_count := 0;

    alt{
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_VALIDATE_REQ := '6078'H}),
                                m_CMN_CMN_CmValidateReq_001()))
                                -> value v_responseMessage {

            if(v_count > 0){
                setverdict(pass,"CM_VALIDATE.REQ message was repeated.",v_count);
            }
            else {
                tc_TT_EVSE_match_session.stop;
            }
            v_count := v_count + 1;
            tc_TT_match_response.start(par_TT_match_response + par_CMN_Transmission_Delay);
            // send invalid CM_VALIDATE.CNF message
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_VALIDATE_CNF := '6079'H}), v_validateCnf))
                              to vc_sut_mac;

            if(v_count > par_C_EV_match_retry) {
              alt{
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_VALIDATE_REQ := '6078'H}),
                                        m_CMN_CMN_CmValidateReq_001())) {
                  setverdict(fail,"CM_VALIDATE.REQ message was repeated, but v_count > " &
                                  "par_C_EV_match_retry.");
                }
                [] tc_TT_match_response.timeout {
                  setverdict(pass,"TT_match_response timeout. " &
                                  "The total number of retries is reached, " &
                                  "the Validation process " &
                                  "shall be considered as FAILED.");
                }
              }
            }
            else{
                repeat;
            }
        }
        [] pt_SLAC_Port.receive {
           setverdict(fail, "Invalid message type or content was received.");
        }
        [] tc_TT_EVSE_match_session.timeout {
           setverdict(fail,"TT_EVSE_match_session timeout. " &
                            "Matching process shall be " &
                            "considered as FAILED.");
            }
        }
        return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmValidate_004(in hexstring v_resultCode, in boolean v_isRepeat)
                                            runs on EVCC_Tester return verdicttype {

        var MME v_requestMessage;

        alt {
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_VALIDATE_REQ := '6078'H}),
                                 md_CMN_CMN_CmValidateReq_002(?)))
                                 -> value v_requestMessage {

               var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                 cm_validate_req.vrVarField.pilot_timer;
               tc_TT_EVSE_match_session.stop;
               if(v_pilotTimer == '00'H) {
                   setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");

                   tc_TT_match_sequence.start(par_TT_match_sequence);

                   pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(v_resultCode)))
                                  to vc_sut_mac;
                   if(v_isRepeat) {
                       repeat;
                   }
               }
               else {
                   tc_TT_match_sequence.stop;
                   setverdict(fail,"Step 2 CM_VALIDATE.REQ was received. The validation " &
                                 "process with the current EVSE has to be stopped by " &
                                 "the SUT before.");
               }
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_REQ := '6078'H}),
                                 mw_CMN_CMN_CmValidateReq_003())) {

               setverdict(fail,"Result field is not set to 'Ready'. Matching process is " &
                               "considered as FAILED.");
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_PARM_REQ := '6064'H}),
                                 md_CMN_CMN_CmSlacParmReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

               setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                 "New Matching process is started.");
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

               // CM_ATTEN_CHAR.RSP messages will be ignored!
               repeat;
            }
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_match_sequence.timeout {
               setverdict(pass,"TT_match_sequence timeout. " &
                               "The SUT has stopped the validation process " &
                               "with the current EVSE.");
            }
            [] tc_TT_EVSE_match_session.timeout {
               setverdict(fail,"TT_EVSE_match_session timeout. Matching process " &
                               "shall be considered as FAILED.");
            }
    }
  return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmValidate_005(in HAL_61851_Listener v_HAL_61851_Listener)
                                          runs on EVCC_Tester return verdicttype {

    var MME v_requestMessage;
    var integer cnt := 0;
    var boolean isStep2 := false;
    var float v_TT_EVSE_vald_toggle;

    alt {
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_REQ := '6078'H}),
                                 md_CMN_CMN_CmValidateReq_002(?)))
                                 -> value v_requestMessage {

            var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                cm_validate_req.vrVarField.pilot_timer;
            tc_TT_EVSE_match_session.stop;
            if(v_pilotTimer == '00'H) {
                if(not isStep2) {
                    setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
                    isStep2 := true;
                }
                else {
                    log("Step 2 CM_VALIDATE.REQ message contains timer field equal to zero. " &
                        "Step 1 CM_VALIDATE.CNF will be resent.");
                }
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(par_cmValidate_result_ready)))
                                  to vc_sut_mac;

                tc_TT_match_sequence.start(par_TT_match_sequence);
                repeat;
            }
            else if(decodeValdToggleTime(v_pilotTimer) >= par_TP_EV_vald_toggle_min and
                    decodeValdToggleTime(v_pilotTimer) <= par_TP_EV_vald_toggle_max and
                    isStep2) {

                setverdict(pass,"Step 2 CM_VALIDATE.REQ is correct.");
                tc_TT_match_sequence.stop;
                v_TT_EVSE_vald_toggle := decodeValdToggleTime(v_pilotTimer);
            }
            else {
                setverdict(fail, "Invalid message content was " &
                                 "received from the SUT.");
            }
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_REQ := '6078'H}),
                                 mw_CMN_CMN_CmValidateReq_003())) {

            setverdict(fail,"Result field is not set to 'Ready'. Matching process is " &
                            "considered as FAILED.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_PARM_REQ := '6064'H}),
                                 md_CMN_CMN_CmSlacParmReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))) {
            setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                              "New Matching process is started.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

            // CM_ATTEN_CHAR.RSP messages will be ignored!
            repeat;
         }
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_match_sequence.timeout {
               cnt := cnt +1;
               log("TT_match_sequence timeout.");
               if (cnt > par_C_EV_match_retry) {
                   setverdict(fail,"Repetition limit is reached. Matching process is " &
                                   "considered as FAILED.");
               }
               else {
                   log("A new CM_VALIDATE.CNF message will be sent.");
                   pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_CNF := '6079'H}),
                                     md_CMN_CMN_CmValidateCnf_001(
                                     par_cmValidate_result_ready)))
                                     to vc_sut_mac;

                   tc_TT_match_sequence.start(par_TT_match_sequence);
                   repeat;
               }
            }
            [] tc_TT_EVSE_match_session.timeout {
               setverdict(fail,"TT_EVSE_match_session timeout. Matching process shall be " &
                                "considered as FAILED.");
            }
        }
        if(getverdict == pass) {
            // BCB toggle sequence detection
            tc_TT_EVSE_vald_toggle.start(v_TT_EVSE_vald_toggle);
            f_EVCC_changeValidStateCondition(B,C);
            timer statetimer := (par_T_vald_state_duration_max + par_CMN_Transmission_Delay);
            statetimer.start;
            var integer toggleCnt := 0;

            alt {
               [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, B){

                   statetimer.stop;
                   toggleCnt := toggleCnt + 1;
                   f_EVCC_changeValidStateCondition(B,C);
                   repeat;
               }
               [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, C){

                   statetimer.stop;
                   f_EVCC_changeValidStateCondition(C,B);
                   statetimer.start(par_T_vald_state_duration_max);
                   repeat;
               }
               [] pt_HAL_61851_Internal_Port.receive {
                   setverdict(fail, "Received state has an invalid value.");
               }
               [] statetimer.timeout {
                   setverdict(fail, "The EVSE could not detect the corresponding " &
                                    "toggle value within " &
                                    "the maximal valid state duration.");
               }
               [] tc_TT_EVSE_vald_toggle.timeout {

                    tc_TT_match_sequence.start(par_TT_match_sequence);
                    alt {
                        [] pt_SLAC_Port.receive {
                            setverdict(fail,"The SUT did not stop the validation " &
                                            "process with the current EVSE.");
                        }
                        [] tc_TT_match_sequence.timeout {
                            setverdict(pass,"TT_match_sequence timeout. " &
                                            "The SUT has stopped the validation " &
                                            "process with the current EVSE.");
                        }
                    }
               }
            }
        }
        return getverdict;
      }

    function f_EVCC_CMN_TB_VTB_CmValidate_006(in HAL_61851_Listener v_HAL_61851_Listener,
                                              in hexstring v_resultCode)
```

```
                                      runs on EVCC_Tester return verdicttype {

        var MME v_requestMessage;
        var integer cnt := 0;
        var boolean isStep2 := false;
        var float v_TT_EVSE_vald_toggle;

        alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_REQ := '6078'H}),
                                     md_CMN_CMN_CmValidateReq_002(?)))
                                     -> value v_requestMessage {

                var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                  cm_validate_req.vrVarField.pilot_timer;
                tc_TT_EVSE_match_session.stop;
                if(v_pilotTimer == '00'H) {
                    if(not isStep2) {
                        setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
                        isStep2 := true;
                    }
                    else {
                        log("Step 2 CM_VALIDATE.REQ message contains timer field equal to zero. " &
                            "Step 1 CM_VALIDATE.CNF will be resent.");
                    }
                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_CNF := '6079'H}),
                                      md_CMN_CMN_CmValidateCnf_001(
                                      par_cmValidate_result_ready)))
                                      to vc_sut_mac;

                    tc_TT_match_sequence.start(par_TT_match_sequence);
                    repeat;
                }
                else if(decodeValdToggleTime(v_pilotTimer) >= par_TP_EV_vald_toggle_min and
                        decodeValdToggleTime(v_pilotTimer) <= par_TP_EV_vald_toggle_max and
                        isStep2) {

                    setverdict(pass,"Step 2 CM_VALIDATE.REQ is correct.");
                    tc_TT_match_sequence.stop;
                    v_TT_EVSE_vald_toggle := decodeValdToggleTime(v_pilotTimer);
                }
                else {
                    setverdict(fail, "Invalid message content was " &
                                     "received from the SUT.");
                }
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_REQ := '6078'H}),
                                     mw_CMN_CMN_CmValidateReq_003())) {

                setverdict(fail,"Result field is not set to 'Ready'. Matching process is " &
                                "considered as FAILED.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_SLAC_PARM_REQ := '6064'H}),
                                     md_CMN_CMN_CmSlacParmReq_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

                setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                  "New Matching process is started.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

                // CM_ATTEN_CHAR.RSP messages will be ignored!
                repeat;
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
             }
             [] tc_TT_match_sequence.timeout {
                cnt := cnt +1;
```

```
            log("TT_match_sequence timeout.");
            if (cnt > par_C_EV_match_retry) {
                setverdict(fail,"Repetition limit is reached. Matching process " &
                                "is considered as FAILED.");
            }
            else {
                log("A new CM_VALIDATE.CNF message will be sent.");
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(
                                  par_cmValidate_result_ready)))
                                  to vc_sut_mac;

                tc_TT_match_sequence.start(par_TT_match_sequence);
                repeat;
            }
         }
         [] tc_TT_EVSE_match_session.timeout {
            setverdict(fail,"TT_EVSE_match_session timeout. Matching process shall " &
                            "be considered as FAILED.");
         }
    }

    if(getverdict == pass) {
        // BCB toggle sequence detection
        tc_TT_EVSE_vald_toggle.start(v_TT_EVSE_vald_toggle);
        f_EVCC_changeValidStateCondition(B,C);
        timer statetimer := (par_T_vald_state_duration_max + par_CMN_Transmission_Delay);
        statetimer.start;
        var integer toggleCnt := 0;

        alt {
            [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, B){

               statetimer.stop;
               toggleCnt := toggleCnt + 1;
               f_EVCC_changeValidStateCondition(B,C);
               repeat;
           }
           [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, C){

               statetimer.stop;
               f_EVCC_changeValidStateCondition(C,B);
               statetimer.start(par_T_vald_state_duration_max);
               repeat;
           }
           [] pt_HAL_61851_Internal_Port.receive {
               setverdict(fail, "Received state has an invalid value.");
           }
           [] statetimer.timeout {
               setverdict(fail, "The EVSE could not detect the corresponding " &
                                 "toggle value within the " &
                                "maximal valid state duration.");
           }
           [] tc_TT_EVSE_vald_toggle.timeout {

            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_VALIDATE_CNF := '6079'H}),
                              md_CMN_CMN_CmValidateCnf_002(
                              int2hex(toggleCnt,2),
                              v_resultCode)))
                              to vc_sut_mac;

            tc_TT_match_sequence.start(par_TT_match_sequence);
                alt {
                    [] pt_SLAC_Port.receive {
                        setverdict(fail,"The SUT did not stop the validation " &
                                        "process with the current EVSE.");
                    }
                    [] tc_TT_match_sequence.timeout {
                        setverdict(pass,"TT_match_sequence timeout. " &
                                        "The SUT has stopped the validation " &
                                        "process with the current EVSE.");
                    }
                }
           }
        }
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmValidate_007() runs on EVCC_Tester return verdicttype {

    var MME v_requestMessage;

    alt {
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_VALIDATE_REQ := '6078'H}),
                                 md_CMN_CMN_CmValidateReq_002(?)))
                                 -> value v_requestMessage {

            var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                cm_validate_req.vrVarField.pilot_timer;
            tc_TT_EVSE_match_session.stop;
            if(v_pilotTimer == '00'H) {
                setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");

                tc_TT_match_sequence.start(par_TT_match_sequence);

                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(
                                  par_cmValidate_result_notRequired)))
                                  to vc_sut_mac;
                repeat;
            }
            else {
                tc_TT_match_sequence.stop;
                setverdict(pass,"Step 2 CM_VALIDATE.REQ was received. SUT has " &
                                "continued the validation process.");
            }
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_VALIDATE_REQ := '6078'H}),
                                 mw_CMN_CMN_CmValidateReq_003())) {

            setverdict(fail,"Result field is not set to 'Ready'. Matching process is " &
                            "considered as FAILED.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_PARM_REQ := '6064'H}),
                                 md_CMN_CMN_CmSlacParmReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

            setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                              "New Matching process is started.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

            // CM_ATTEN_CHAR.RSP messages will be ignored!
            repeat;
         }
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
         }
         [] tc_TT_match_sequence.timeout {
            setverdict(fail,"TT_match_sequence timeout. The SUT has stopped " &
                            "the validation process.");
         }
         [] tc_TT_EVSE_match_session.timeout {
            setverdict(fail,"TT_EVSE_match_session timeout. Matching process " &
                            "shall be considered as FAILED.");
         }
    }
  return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmValidate_008() runs on EVCC_Tester return verdicttype {
        var MME v_requestMessage;
        var boolean v_isFirst := true;

        alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_REQ := '6078'H}),
                                     md_CMN_CMN_CmValidateReq_002('00'H)))
                                     -> value v_requestMessage {

                if(v_isFirst) {
                    tc_TT_EVSE_match_session.stop;
                    setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_CNF := '6079'H}),
                                      md_CMN_CMN_CmValidateCnf_001(
                                      par_cmValidate_result_notReady)))
                                      to vc_sut_mac;
                }
                else {
                    tc_EVCC_ValidationRetry.stop;
                    setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct. SUT has retried " &
                                    "the SLAC validation process after waiting for " &
                                    "PIXIT_EVCC_CMN_ValidationRetry seconds.");
                }
                if(v_isFirst) {
                   tc_EVCC_ValidationRetry.start(PIXIT_EVCC_CMN_ValidationRetry +
                                                 (2.0 * par_CMN_Transmission_Delay));
                   v_isFirst := false;
                   repeat;
                }
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_REQ := '6078'H}),
                                     mw_CMN_CMN_CmValidateReq_003())) {

                setverdict(fail,"Result field is not set to 'Ready'. Matching process is " &
                                "considered as FAILED.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_PARM_REQ := '6064'H}),
                                     md_CMN_CMN_CmSlacParmReq_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

                setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                  "New Matching process is started.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

                // CM_ATTEN_CHAR.RSP messages will be ignored!
                repeat;
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
             }
             [] tc_TT_EVSE_match_session.timeout {
                setverdict(fail,"TT_EVSE_match_session timeout. Matching process " &
                                "shall be considered as FAILED.");
             }
             [] tc_EVCC_ValidationRetry.timeout {
                setverdict(fail,"EVCC_ValidationRetry timeout. SUT has not retried " &
                                "the SLAC validation process after waiting for " &
                                "'PIXIT_EVCC_CMN_ValidationRetry' seconds.");
             }
        }
      return getverdict;
    }

    // SLAC Tester

    function f_EVCC_CMN_TB_VTB_CmValidate_009() runs on SLAC_Tester return verdicttype {

        var MME v_responseMessage;
        var integer v_count;
    var verdicttype v_verdict;

    v_verdict := f_EVCC_CMN_TB_VTB_CmValidatePreCondition_001();

    if(v_verdict == pass) {
        v_count := 0;
        alt{
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_VALIDATE_REQ := '6078'H}),
                                    m_CMN_CMN_CmValidateReq_001()))
                                    -> value v_responseMessage {

                if(v_count > 0){
                    setverdict(pass,"CM_VALIDATE.REQ message was repeated.",v_count);
                }
                else {
                    tc_TT_EVSE_match_session.stop;
                }
                v_count := v_count + 1;
                tc_TT_match_response.start(par_TT_match_response +
                                           par_CMN_Transmission_Delay);

                if(v_count > par_C_EV_match_retry) {
                  alt{
                    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_VALIDATE_REQ := '6078'H}),
                                            m_CMN_CMN_CmValidateReq_001())) {
                      setverdict(fail,"CM_VALIDATE.REQ message " &
                                        "was repeated, but v_count > " &
                                      "par_C_EV_match_retry.");
                    }
                    [] tc_TT_match_response.timeout {
                      setverdict(pass,"TT_match_response timeout. " &
                                        "The total number of retries is reached, " &
                                      "the Validation process " &
                                      "shall be considered as FAILED.");
                     }
                  }
                }
                else{
                    repeat;
                }
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
             }
             [] tc_TT_EVSE_match_session.timeout {
                setverdict(fail,"TT_EVSE_match_session timeout. " &
                                "Matching process shall be " &
                                "considered as FAILED.");
             }
         }
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmValidate_010(in hexstring v_resultCode)
                                          runs on SLAC_Tester return verdicttype {

    var MME v_requestMessage;
    var verdicttype v_verdict;

    v_verdict := f_EVCC_CMN_TB_VTB_CmValidatePreCondition_001();

    if(v_verdict == pass) {

        alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_REQ := '6078'H}),
                                     md_CMN_CMN_CmValidateReq_002(?)))
                                     -> value v_requestMessage {

                var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                    cm_validate_req.vrVarField.pilot_timer;
                tc_TT_EVSE_match_session.stop;
                if(v_pilotTimer == '00'H) {
                    setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");

                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_CNF := '6079'H}),
                                      md_CMN_CMN_CmValidateCnf_001(v_resultCode)))
                                      to vc_sut_mac;

                    tc_TT_match_sequence.start(par_TT_match_sequence);
                    repeat;
                }
                else {
                    tc_TT_match_sequence.stop;
                    setverdict(fail,"Step 2 CM_VALIDATE.REQ was received. The validation " &
                                    "process with the current EVSE has to be stopped by " &
                                    "the SUT before.");
                }
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_REQ := '6078'H}),
                                     mw_CMN_CMN_CmValidateReq_003())) {

                setverdict(fail,"Result field is not set to 'Ready'. Matching process is " &
                                "considered as FAILED.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_PARM_REQ := '6064'H}),
                                     md_CMN_CMN_CmSlacParmReq_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

                setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                  "New Matching process is started.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

                // CM_ATTEN_CHAR.RSP messages will be ignored!
                repeat;
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
             }
             [] tc_TT_match_sequence.timeout {
                setverdict(pass,"TT_match_sequence timeout. " &
                                "The SUT has stopped the validation process " &
                                "with the current EVSE.");
             }
             [] tc_TT_EVSE_match_session.timeout {
                setverdict(fail,"TT_EVSE_match_session timeout. Matching process " &
                                "shall be considered as FAILED.");
             }
         }
   }
   return getverdict;
  }

  function f_EVCC_CMN_TB_VTB_CmValidate_011(in HAL_61851_Listener v_HAL_61851_Listener)
                                            runs on SLAC_Tester return verdicttype {

      var MME v_requestMessage;
      var integer cnt := 0;
      var boolean isStep2 := false;
      var float v_TT_EVSE_vald_toggle;
      var verdicttype v_verdict;

      v_verdict := f_EVCC_CMN_TB_VTB_CmValidatePreCondition_001();

      if(v_verdict == pass) {

          alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_REQ := '6078'H}),
                                     md_CMN_CMN_CmValidateReq_002(?)))
                                     -> value v_requestMessage {
```

```
   var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                       cm_validate_req.vrVarField.pilot_timer;
   tc_TT_EVSE_match_session.stop;
   if(v_pilotTimer == '00'H) {
       if(not isStep2) {
           setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
           isStep2 := true;
       }
       else {
           log("Step 2 CM_VALIDATE.REQ message contains " &
               "timer field equal to zero. " &
               "Step 1 CM_VALIDATE.CNF will be resent.");
       }
       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                         md_CMN_CMN_SlacMmeCmnHeader_001({
                         CM_VALIDATE_CNF := '6079'H}),
                         md_CMN_CMN_CmValidateCnf_001(
                         par_cmValidate_result_ready)))
                         to vc_sut_mac;

       tc_TT_match_sequence.start(par_TT_match_sequence);
       repeat;
   }
   else if(decodeValdToggleTime(v_pilotTimer) >= par_TP_EV_vald_toggle_min and
           decodeValdToggleTime(v_pilotTimer) <= par_TP_EV_vald_toggle_max and
           isStep2) {

       setverdict(pass,"Step 2 CM_VALIDATE.REQ is correct.");
       tc_TT_match_sequence.stop;
       v_TT_EVSE_vald_toggle := decodeValdToggleTime(v_pilotTimer);
   }
   else {
       setverdict(fail, "Invalid message content was " &
                        "received from the SUT.");
   }
}
[] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                        md_CMN_CMN_SlacMmeCmnHeader_001({
                        CM_VALIDATE_REQ := '6078'H}),
                        mw_CMN_CMN_CmValidateReq_003())) {

   setverdict(fail,"Result field is not set to 'Ready'. " &
                   "Matching process is " &
                   "considered as FAILED.");
}
[] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                        md_CMN_CMN_SlacMmeCmnHeader_001({
                        CM_SLAC_PARM_REQ := '6064'H}),
                        md_CMN_CMN_CmSlacParmReq_001(
                        m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

   setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                     "New Matching process is started.");
}
[] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                        md_CMN_CMN_SlacMmeCmnHeader_001({
                        CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

   // CM_ATTEN_CHAR.RSP messages will be ignored!
   repeat;
}
[] pt_SLAC_Port.receive {
   setverdict(fail, "Invalid message type or content was received.");
}
[] tc_TT_match_sequence.timeout {
   cnt := cnt +1;
   log("TT_match_sequence timeout.");
   if (cnt > par_C_EV_match_retry) {
       setverdict(fail,"Repetition limit is reached. " &
                       "Matching process is " &
                       "considered as FAILED.");
   }
   else {
       log("A new CM_VALIDATE.CNF message will be sent.");
       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                         md_CMN_CMN_SlacMmeCmnHeader_001({
                         CM_VALIDATE_CNF := '6079'H}),
                                    md_CMN_CMN_CmValidateCnf_001(
                                    par_cmValidate_result_ready)))
                                    to vc_sut_mac;

                    tc_TT_match_sequence.start(par_TT_match_sequence);
                    repeat;
                }
             }
             [] tc_TT_EVSE_match_session.timeout {
                setverdict(fail,"TT_EVSE_match_session timeout. " &
                                "Matching process shall be " &
                                "considered as FAILED.");
             }
        }
        if(getverdict == pass) {
            // BCB toggle sequence detection
            tc_TT_EVSE_vald_toggle.start(v_TT_EVSE_vald_toggle);
            f_EVCC_changeValidStateCondition(B,C);
            timer statetimer := (par_T_vald_state_duration_max +
                                 par_CMN_Transmission_Delay);
            statetimer.start;
            var integer toggleCnt := 0;

            alt {
                [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, B){

                    statetimer.stop;
                    toggleCnt := toggleCnt + 1;
                    f_EVCC_changeValidStateCondition(B,C);
                    repeat;
               }
               [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, C){

                    statetimer.stop;
                    f_EVCC_changeValidStateCondition(C,B);
                    statetimer.start(par_T_vald_state_duration_max);
                    repeat;
               }
               [] pt_HAL_61851_Internal_Port.receive {
                    setverdict(fail, "Received state has an invalid value.");
               }
               [] statetimer.timeout {
                    setverdict(fail, "The EVSE could not detect the corresponding " &
                                     "toggle value within " &
                                     "the maximal valid state duration.");
               }
               [] tc_TT_EVSE_vald_toggle.timeout {

                     tc_TT_match_sequence.start(par_TT_match_sequence);
                     alt {
                         [] pt_SLAC_Port.receive {
                             setverdict(fail,"The SUT did not stop the validation " &
                                             "process with the current EVSE.");
                         }
                         [] tc_TT_match_sequence.timeout {
                             setverdict(pass,"TT_match_sequence timeout. " &
                                             "The SUT has stopped the validation " &
                                             "process with the current EVSE.");
                         }
                     }
               }
            }
        }
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmValidate_012(in HAL_61851_Listener v_HAL_61851_Listener,
                                          in hexstring v_resultCode)
                                          runs on SLAC_Tester return verdicttype {

    var MME v_requestMessage;
    var integer cnt := 0;
    var boolean isStep2 := false;
    var float v_TT_EVSE_vald_toggle;
    var verdicttype v_verdict;

    v_verdict := f_EVCC_CMN_TB_VTB_CmValidatePreCondition_001();
```

```
if(v_verdict == pass) {

    alt {
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_VALIDATE_REQ := '6078'H}),
                                 md_CMN_CMN_CmValidateReq_002(?)))
                                 -> value v_requestMessage {

            var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                               cm_validate_req.vrVarField.pilot_timer;
            tc_TT_EVSE_match_session.stop;
            if(v_pilotTimer == '00'H) {
                if(not isStep2) {
                    setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
                    isStep2 := true;
                }
                else {
                    log("Step 2 CM_VALIDATE.REQ message contains " &
                        "timer field equal to zero. " &
                        "Step 1 CM_VALIDATE.CNF will be resent.");
                }
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(
                                  par_cmValidate_result_ready)))
                                  to vc_sut_mac;

                tc_TT_match_sequence.start(par_TT_match_sequence);
                repeat;
            }
            else if(decodeValdToggleTime(v_pilotTimer) >= par_TP_EV_vald_toggle_min and
                    decodeValdToggleTime(v_pilotTimer) <= par_TP_EV_vald_toggle_max and
                    isStep2) {

                setverdict(pass,"Step 2 CM_VALIDATE.REQ is correct.");
                tc_TT_match_sequence.stop;
                v_TT_EVSE_vald_toggle := decodeValdToggleTime(v_pilotTimer);
            }
            else {
                setverdict(fail, "Invalid message content was " &
                                 "received from the SUT.");
            }
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_VALIDATE_REQ := '6078'H}),
                                 mw_CMN_CMN_CmValidateReq_003())) {

            setverdict(fail,"Result field is not set to 'Ready'. " &
                            "Matching process is " &
                            "considered as FAILED.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_PARM_REQ := '6064'H}),
                                 md_CMN_CMN_CmSlacParmReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

            setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                              "New Matching process is started.");
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

            // CM_ATTEN_CHAR.RSP messages will be ignored!
            repeat;
         }
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
         }
         [] tc_TT_match_sequence.timeout {
            cnt := cnt +1;
            log("TT_match_sequence timeout.");
            if (cnt > par_C_EV_match_retry) {
                setverdict(fail,"Repetition limit is reached. " &
                                "Matching process " &
                                "is considered as FAILED.");
                }
                else {
                    log("A new CM_VALIDATE.CNF message will be sent.");
                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_CNF := '6079'H}),
                                      md_CMN_CMN_CmValidateCnf_001(
                                      par_cmValidate_result_ready)))
                                      to vc_sut_mac;

                    tc_TT_match_sequence.start(par_TT_match_sequence);
                    repeat;
                }
            }
            [] tc_TT_EVSE_match_session.timeout {
                setverdict(fail,"TT_EVSE_match_session timeout. " &
                                "Matching process shall " &
                                "be considered as FAILED.");
            }
        }

        if(getverdict == pass) {
            // BCB toggle sequence detection
            tc_TT_EVSE_vald_toggle.start(v_TT_EVSE_vald_toggle);
            f_EVCC_changeValidStateCondition(B,C);
            timer statetimer := (par_T_vald_state_duration_max +
                                 par_CMN_Transmission_Delay);
            statetimer.start;
            var integer toggleCnt := 0;

            alt {
                [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, B){

                    statetimer.stop;
                    toggleCnt := toggleCnt + 1;
                    f_EVCC_changeValidStateCondition(B,C);
                    repeat;
                }
                [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, C){

                    statetimer.stop;
                    f_EVCC_changeValidStateCondition(C,B);
                    statetimer.start(par_T_vald_state_duration_max);
                    repeat;
                }
                [] pt_HAL_61851_Internal_Port.receive {
                    setverdict(fail, "Received state has an invalid value.");
                }
                [] statetimer.timeout {
                    setverdict(fail, "The EVSE could not detect the corresponding " &
                                    "toggle value within the " &
                                    "maximal valid state duration.");
                }
                [] tc_TT_EVSE_vald_toggle.timeout {

                   pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_VALIDATE_CNF := '6079'H}),
                                    md_CMN_CMN_CmValidateCnf_002(
                                    int2hex(toggleCnt,2),
                                    v_resultCode)))
                                    to vc_sut_mac;

                   tc_TT_match_sequence.start(par_TT_match_sequence);
                     alt {
                         [] pt_SLAC_Port.receive {
                             setverdict(fail,"The SUT did not stop the validation " &
                                             "process with the current EVSE.");
                         }
                         [] tc_TT_match_sequence.timeout {
                             setverdict(pass,"TT_match_sequence timeout. " &
                                             "The SUT has stopped the validation " &
                                             "process with the current EVSE.");
                         }
                     }
                }
            }
            }
        }
        return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmValidate_013(in HAL_61851_PwmMode_Type pwmMode)
                                          runs on EVCC_Tester return verdicttype {

        var MME v_requestMessage;
        var boolean isStep2 := false;

        alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_REQ := '6078'H}),
                                     md_CMN_CMN_CmValidateReq_002(?)))
                                     -> value v_requestMessage {

                var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                    cm_validate_req.vrVarField.pilot_timer;
                tc_TT_EVSE_match_session.stop;
                if(v_pilotTimer == '00'H) {
                    if(not isStep2) {
                        setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
                        isStep2 := true;
                    }
                    else {
                        setverdict(fail,"Step 2 CM_VALIDATE.REQ message contains " &
                                        "timer field equal to zero and was not expected. " &
                                        "CP State E/F should be detected before.");
                        break;
                    }

                    // set error state
                        f_EVCC_changeValidStateCondition(E,E);
                        f_EVCC_setPwmMode(pwmMode);


                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_VALIDATE_CNF := '6079'H}),
                                      md_CMN_CMN_CmValidateCnf_001(
                                      par_cmValidate_result_ready)))
                                      to vc_sut_mac;

                    tc_TT_match_sequence.start(par_TT_match_sequence);
                    repeat;
                }
                else {
                    setverdict(fail,"Step 2 CM_VALIDATE.REQ message was not expected. " &
                                    "CP State E/F should be detected before.");
                }
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_VALIDATE_REQ := '6078'H}),
                                     mw_CMN_CMN_CmValidateReq_003())) {

                    setverdict(fail,"Result field is not set to 'Ready'. Matching process is " &
                                    "considered as FAILED.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_PARM_REQ := '6064'H}),
                                     md_CMN_CMN_CmSlacParmReq_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), ?))){

                setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                  "New Matching process is started.");
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_ATTEN_CHAR_RSP := '606F'H}),?)) {
                // CM_ATTEN_CHAR.RSP messages will be ignored!
                repeat;
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
```

```
           }
           [] tc_TT_match_sequence.timeout {
              setverdict(pass,"TT_match_sequence timer has expired, " &
                              "the Matching process was terminated " &
                              "by the SUT.");
           }
           [] tc_TT_EVSE_match_session.timeout {
              setverdict(fail,"TT_EVSE_match_session timeout. Matching process shall " &
                              "be considered as FAILED.");
           }
     }
     return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmValidatePreCondition_001() runs on SLAC_Tester
                                                         return verdicttype {
     var MME v_responseMessage;
     var integer v_Num_soundsInt;
     var ResponseMessageList_TYPE reponseMessageList;
     var hexstring  v_variable;
     var integer v_count := 0;
     var boolean v_isRunning := true;
     var boolean v_repetition := true;
     var AttenProfile_TYPE v_attenuation_list;
     var integer v_count2 := 0;
     var integer v_countDecrement;
     var integer v_countStart;
     var integer v_countStop;
     var boolean v_firstSound := true;

    alt{
         []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                               md_CMN_CMN_SlacMmeCmnHeader_001({
                               CM_SLAC_PARM_REQ := '6064'H}),
                               md_CMN_CMN_CmSlacParmReq_001(
                               m_CMN_CMN_SlacPayloadHeader_001(), ?)))
                               -> value v_responseMessage {

            setverdict(pass,"CM_SLAC_PARM.REQ is correct.");
            vc_RunID := v_responseMessage.mme_payload.payload.cm_slac_parm_req.runid;
            tc_TT_match_sequence.start(par_TT_match_sequence);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_SLAC_PARM_CNF := '6065'H}),
                             md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                             m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                             to vc_sut_mac;
         }
         [] a_EVCC_processPLCLinkNotifications_002();
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
         }
         [] tc_TT_EVSE_SLAC_init.timeout {
            setverdict(fail,"TT_EVSE_SLAC_init timeout. SECC assumes that no " &
                           "SLAC will be performed.");
           }
     }

     alt {
          [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_START_ATTEN_CHAR_IND := '606A'H}),
                                  md_CMN_CMN_CmStartAttenCharInd_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), ?, ?,
                                  '01'H, ?, vc_RunID)))
                                  -> value v_responseMessage {

              if(v_count2 == 0) {
                 tc_TT_EVSE_match_MNBC.start(par_TT_EVSE_match_MNBC);
                 tc_TT_match_sequence.stop;

                 vc_Num_sounds := v_responseMessage.mme_payload.payload.
                                  cm_start_atten_char_ind.num_sounds;
                 v_Num_soundsInt := hex2int(vc_Num_sounds);
              }
              setverdict(pass,"CM_START_ATTEN_CHAR.IND is correct.");
              v_count2 := v_count2 + 1;
              if(v_count2 < cc_numberOfStartAtten) {
                 repeat;
            }
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_SLAC_PARM_REQ := '6064'H}),
                                md_CMN_CMN_CmSlacParmReq_001(
                                m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

            tc_TT_match_sequence.start(par_TT_match_sequence);
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_SLAC_PARM_CNF := '6065'H}),
                             md_CMN_CMN_CmSlacParmCnf_001(vc_sut_mac,
                             m_CMN_CMN_SlacPayloadHeader_001(), vc_RunID)))
                             to vc_sut_mac;

            log("A further CM_SLAC_PARM.REQ message was received. " &
                "A new CM_SLAC_PARM.CNF has to be send.");
            repeat;
         }
         [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or content was received.");
            if(v_count2 < cc_numberOfStartAtten) {
                setverdict(fail, "A wrong number of CM_START_ATTEN_CHAR.IND " &
                                 "message was received.");
            }
         }
         [] tc_TT_match_sequence.timeout {
            setverdict(fail,"TT_match_sequence timeout. " &
                             "No CM_START_ATTEN_CHAR.IND " &
                            "message was received. Matching process shall be " &
                            "considered as FAILED.");
            break;
         }
         [] tc_TT_EVSE_match_MNBC.timeout {
            setverdict(fail,"TT_EVSE_match_MNBC timeout. A wrong number of " &
                            "CM_START_ATTEN_CHAR.IND message was received. " &
                            "Matching process shall be considered as FAILED.");
         }
    }
   if(getverdict == pass) {
       while (v_isRunning) {
           alt {
                 [v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_MNBC_SOUND_IND := '6076'H}),
                                      md_CMN_CMN_CmMnbcSoundInd_001(
                                      m_CMN_CMN_SlacPayloadHeader_001(),
                                      ?, vc_RunID, ?)))
                                      -> value v_responseMessage {

                    v_countStart := hex2int(v_responseMessage.
                                            mme_payload.payload.
                                            cm_mnbc_sound_ind.count);
                    v_firstSound := false;
                    if(v_countStart == cc_numberOfSoundings) {
                        v_countDecrement := v_countStart - 1;
                        v_countStop := 1;
                    } else if(v_countStart == cc_numberOfSoundings - 1){
                        v_countDecrement := v_countStart - 1;
                        v_countStop := 0;
                    }
                    else {setverdict(fail, "The field 'count' has an " &
                                           "invalid value.");
                       v_isRunning := false;
                       break;
                    }
                    repeat;
                 }
                 [not v_firstSound] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_MNBC_SOUND_IND := '6076'H}),
                                    md_CMN_CMN_CmMnbcSoundInd_001(
                                    m_CMN_CMN_SlacPayloadHeader_001(),
                                    int2hex(v_countDecrement,2),
                                    vc_RunID, ?))) {
                    v_countDecrement := v_countDecrement - 1;
                    repeat;
                }
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_ATTEN_PROFILE_IND := '6086'H}),
                                        md_EVCC_CMN_CmAttenProfileInd_001(
                                        vc_sut_mac, ?, *)))
                                        -> value v_responseMessage {

                  if(ispresent(v_responseMessage.mme_payload.payload.
                               cm_atten_profile_ind.attenuation_list)) {

                      if(v_responseMessage.mme_payload.payload.
                         cm_atten_profile_ind.num_groups != '3A'H) {
                          setverdict(fail,"Invalid numGroups value detected.");
                          v_isRunning := false;
                          break;
                      }
                      reponseMessageList[v_count] := v_responseMessage;
                      v_count := v_count + 1;
                  }
                  else {
                      log("Attenuation list was empty, the received message could not " &
                          "be considered for attenuation calculation.");
                  }
                }
                [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                        md_CMN_CMN_SlacMmeCmnHeader_001({
                                        CM_SLAC_PARM_REQ := '6064'H}),
                                        md_CMN_CMN_CmSlacParmReq_001(
                                        m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

                   setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                     "New Matching process is started.");
                   v_isRunning := false;
                }
                [] pt_SLAC_Port.receive {
                   setverdict(fail, "Invalid message type or content was received.");
                   v_isRunning := false;
                }
                [] tc_TT_EVSE_match_MNBC.timeout {
                   v_isRunning := false;
                 }
              }
          if(v_count == v_Num_soundsInt){
              tc_TT_EVSE_match_MNBC.stop;
              v_isRunning := false;
          }
       }
    }
    if (v_count>0){
        if(v_countDecrement != (v_countStop - 1)) {
           setverdict(fail,"A wrong number of CM_MNBC_SOUND.IND messages " &
                           "was received.");
        } else {
           vc_Num_sounds := int2hex(v_count,2);
           setverdict(pass,"CM_MNBC_SOUND.IND is correct.");
           setverdict(pass,"CM_ATTEN_PROFILE.IND is correct.");
           v_attenuation_list := m_EVCC_CMN_atten_list_002();
        }
    }
    else {
        setverdict(fail,"No Atten Profile messages received.");
    }

    if(getverdict == pass) {
        v_count := 0;
        while(v_repetition){

            tc_TT_match_response.start(par_TT_match_response);
            v_count := v_count + 1;
            pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_ATTEN_CHAR_IND    := '606E'H}),
                              md_EVCC_CMN_CmAttenCharInd_001(
                              m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                              vc_RunID, vc_Num_sounds, v_attenuation_list)))
                              to vc_sut_mac;
             alt {
                  [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_ATTEN_CHAR_RSP :='606F'H}),
                                     md_CMN_CMN_CmAttenCharRsp_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(),
                                     md_CMN_CMN_Acvarfield_001(
                                     vc_sut_mac, vc_RunID)))) {

                        setverdict(pass,"CM_ATTEN_CHAR.RSP is correct.");
                        v_repetition := false;
                        tc_TT_match_response.stop;
                        tc_TT_EVSE_match_session.start(par_TT_EVSE_match_session);
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_PARM_REQ := '6064'H}),
                                     md_CMN_CMN_CmSlacParmReq_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

                        setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                     "New Matching process is started.");
                        v_repetition := false;
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_MNBC_SOUND_IND := '6076'H}),?)) {
                        // CM_ATTEN_PROFILE.IND messages will be ignored!
                        repeat;
                     }
                     [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_ATTEN_PROFILE_IND := '6086'H}),?)) {
                        // CM_ATTEN_PROFILE.IND messages will be ignored!
                        repeat;
                     }
                     [] pt_SLAC_Port.receive {
                        setverdict(fail, "Invalid message type or content was received.");
                        v_repetition := false;
                     }
                     [] tc_TT_match_response.timeout {
                         log("TT_match_response timeout.");
                         if(v_count mod (par_C_EV_match_retry+1) == 0){
                             setverdict(fail,"The repetition limit is reached. " &
                                     "The Matching process is considered " &
                                     "as FAILED.");
                             v_repetition := false;
                         } else {
                           log("The repetition limit is not reached, " &
                               "a new CM_ATTEN_CHAR.IND message will be send.");
                         }
                   }
              }
          }
      }
      return getverdict;
  }

  function decodeValdToggleTime(in PilotTimer_TYPE v_pilotTimer) return float {

      return (int2float((hex2int(v_pilotTimer) + 1)) * 0.1);

  }
}
```

E.2.4 EVCC functions for CmValidateOrCmSlacMatch

```
module TestBehavior_EVCC_CmValidateOrCmSlacMatch {

    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Templates_CMN_CmValidate all;
    import from Templates_CMN_CmStartAttenCharInd all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmMnbcSoundInd all;
    import from Templates_CMN_CmAttenCharRsp all;
    import from Templates_EVCC_CmAttenCharInd all;
    import from Templates_EVCC_CmAttenProfileInd all;
import from Templates_CMN_CmSlacParm all;
import from ComponentsAndPorts all;
import from DataStructure_SLAC all;
import from TestBehavior_EVCC_AttenuationCharacterization all;
import from TestBehavior_EVCC_CmValidate all;
import from LibFunctions_15118_3  { group generalFunctions; }
import from Services_HAL_61851 all;
import from Templates_CMN_CmSlacParm all;
import from Templates_CMN_HAL61851 all;
import from DataStructure_HAL_61851 all;
import from Templates_CMN_CmSlacMatch all;
import from Timer_15118 all;

function f_EVCC_CMN_TB_VTB_CmValidateOrCmSlacMatch_001(in HAL_61851_Listener
                                                       v_HAL_61851_Listener,
                                                       in verdicttype v_vct)
                                                       runs on EVCC_Tester
                                                       return verdicttype {

    var MME v_requestMessage;
    var integer cnt := 0;
    var boolean isStep2 := false;
    var float v_TT_EVSE_vald_toggle;
    var boolean isCmValidate := false;

    alt {
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_MATCH_REQ := '607C'H}),
                                 md_CMN_CMN_CmSlacMatchReq_001(
                                 m_CMN_CMN_SlacPayloadHeader_001(),
                                 vc_sut_mac, par_testSystem_mac,
                                 vc_RunID))) {

            setverdict(pass,"CM_SLAC_MATCH.REQ is correct.");
            tc_TT_EVSE_match_session.stop;
         }
         [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_REQ := '6078'H}),
                                 md_CMN_CMN_CmValidateReq_004(?,?,?)))
                                 -> value v_requestMessage {

            var PilotTimer_TYPE v_pilotTimer := v_requestMessage.mme_payload.payload.
                                                cm_validate_req.vrVarField.pilot_timer;
            var SignalType_TYPE p_signalType := v_requestMessage.mme_payload.payload.
                                                cm_validate_req.signalType;
            if((p_signalType != '00'H)) {
                setverdict(v_vct,"Step 1 CM_VALIDATE.REQ is not correct. Invalid " &
                                 "signalType was detected.");
            }
            tc_TT_EVSE_match_session.stop;
            if(v_pilotTimer == '00'H) {
                if(not isStep2) {
                    setverdict(pass,"Step 1 CM_VALIDATE.REQ is correct.");
                    isStep2 := true;
                }
                else {
                    log("Step 2 CM_VALIDATE.REQ message contains timer field equal to zero. " &
                        "Step 1 CM_VALIDATE.CNF will be resent.");
                }
                pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_VALIDATE_CNF := '6079'H}),
                                  md_CMN_CMN_CmValidateCnf_001(
                                  par_cmValidate_result_ready)))
                                  to vc_sut_mac;

                tc_TT_match_sequence.start(par_TT_match_sequence);
                repeat;
            }
            else if(decodeValdToggleTime(v_pilotTimer) >= par_TP_EV_vald_toggle_min and
                    decodeValdToggleTime(v_pilotTimer) <= par_TP_EV_vald_toggle_max and
                    isStep2) {

                setverdict(pass,"Step 2 CM_VALIDATE.REQ is correct.");
                tc_TT_match_sequence.stop;
                v_TT_EVSE_vald_toggle := decodeValdToggleTime(v_pilotTimer);
            // BCB toggle sequence detection
            tc_TT_EVSE_vald_toggle.start(v_TT_EVSE_vald_toggle);
            f_EVCC_changeValidStateCondition(B,C);
            timer statetimer := (par_T_vald_state_duration_max + par_CMN_Transmission_Delay);
            statetimer.start;
            var integer toggleCnt := 0;

            alt {
                [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, B){

                    statetimer.stop;
                    toggleCnt := toggleCnt + 1;
                    f_EVCC_changeValidStateCondition(B,C);
                    repeat;
                }
                [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, C){

                    statetimer.stop;
                    f_EVCC_changeValidStateCondition(C,B);
                    statetimer.start(par_T_vald_state_duration_max);
                    repeat;
                }
                [] pt_HAL_61851_Internal_Port.receive {
                    setverdict(v_vct, "Received state has an invalid value.");
                }
                [] statetimer.timeout {
                    setverdict(v_vct, "The EVSE could not detect the corresponding " &
                                      "toggle value within the " &
                                      "maximal valid state duration.");
                }
                [] tc_TT_EVSE_vald_toggle.timeout {

                    if(toggleCnt > 0 and toggleCnt <= 3) {
                        setverdict(pass,"Valid BCB toggle sequence could be detected.");
                        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                            md_CMN_CMN_SlacMmeCmnHeader_001({
                                            CM_VALIDATE_CNF := '6079'H}),
                                          md_CMN_CMN_CmValidateCnf_002(
                                          int2hex(toggleCnt,2),
                                          par_cmValidate_result_success)))
                                          to vc_sut_mac;

                        f_EVCC_changeValidStateCondition(C,B);
                        tc_TT_match_sequence.start(par_TT_match_sequence);
                    }
                    else {
                       setverdict(v_vct,"Invalid BCB toggle sequence was detected.");
                    }
                }
            }
            if(getverdict == pass) {
                repeat;
            }
        }
        else {
            setverdict(v_vct, "Invalid message content was " &
                              "received from the SUT.");
        }
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_VALIDATE_REQ := '6078'H}),
                            mw_CMN_CMN_CmValidateReq_003())) {

            setverdict(v_vct,"Result field is not set to 'Ready'. " &
                             "Matching process is " &
                             "considered as FAILED.");
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_SLAC_PARM_REQ := '6064'H}),
                            md_CMN_CMN_CmSlacParmReq_001(
                            m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

        setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                         "New Matching process is started.");
    }
    [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

                // CM_ATTEN_CHAR.RSP messages will be ignored!
                repeat;
             }
             [] pt_SLAC_Port.receive {
                setverdict(v_vct, "Invalid message type or content was received.");
             }
             [] tc_TT_match_sequence.timeout {
                if(isCmValidate) {
                    cnt := cnt +1;
                    log("TT_match_sequence timeout.");
                    if (cnt > par_C_EV_match_retry) {
                        setverdict(v_vct,"Repetition limit is reached. " &
                                         "Matching process is " &
                                         "considered as FAILED.");
                    }
                    else {
                        log("A new CM_VALIDATE.CNF message will be sent.");
                        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_VALIDATE_CNF := '6079'H}),
                                          md_CMN_CMN_CmValidateCnf_001(
                                          par_cmValidate_result_ready)))
                                          to vc_sut_mac;

                        tc_TT_match_sequence.start(par_TT_match_sequence);
                        repeat;
                    }
                }
                else {
                    setverdict(v_vct,"TT_match_sequence timeout. Matching process " &
                                     "shall be considered as FAILED.");
                }
             }
             [] tc_TT_EVSE_match_session.timeout {
                setverdict(v_vct,"TT_EVSE_match_session timeout. Matching process " &
                                 "shall be considered as FAILED.");
             }
        }
        return getverdict;
      }
}
```

E.2.5 EVCC functions for CmSlacMatch

```
module TestBehavior_EVCC_CmSlacMatch {

    import from Templates_CMN_CmSlacMatch all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from ComponentsAndPorts all;
    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from DataStructure_SLAC all;
    import from Templates_CMN_CmSlacParm all;
    import from Timer_15118 all;

    function f_EVCC_CMN_TB_VTB_CmSlacMatch_001() runs on EVCC_Tester return verdicttype {

        alt {
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_SLAC_MATCH_REQ := '607C'H}),
                               md_CMN_CMN_CmSlacMatchReq_001(
                               m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                               par_testSystem_mac, vc_RunID))) {

                setverdict(pass,"CM_SLAC_MATCH.REQ is correct.");
                if(tc_TT_EVSE_match_session.running) {
                    tc_TT_EVSE_match_session.stop;
                }
                else {
                    tc_TT_match_sequence.stop;
                }
             }
             [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                               md_CMN_CMN_SlacMmeCmnHeader_001({
                               CM_SLAC_PARM_REQ := '6064'H}),
                              md_CMN_CMN_CmSlacParmReq_001(
                              m_CMN_CMN_SlacPayloadHeader_001(), ?))) {

               setverdict(inconc,"CM_SLAC_PARM.REQ message was received. " &
                                 "New Matching process is started.");
            }
            [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                               md_CMN_CMN_SlacMmeCmnHeader_001({
                               CM_ATTEN_CHAR_RSP := '606F'H}),?)) {

               // CM_ATTEN_CHAR.RSP messages will be ignored!
               repeat;
            }
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
            [] tc_TT_EVSE_match_session.timeout {
               setverdict(fail,"TT_EVSE_match_session timeout. Matching process shall " &
                                "be considered as FAILED.");
            }
            [] tc_TT_match_sequence.timeout {
               setverdict(fail,"TT_match_sequence timeout. Matching process shall " &
                               "be considered as FAILED.");
            }
        }
        return getverdict;
   }

   function f_EVCC_CMN_TB_VTB_CmSlacMatch_002() runs on EVCC_Tester return verdicttype {

       var MME v_responseMessage;
       var MACAddress_TYPE v_address;
       var integer v_count := 0;
       var boolean v_stopLnkStatus := false;

       v_count := v_count + 1;
       tc_TT_match_response.start(par_TT_match_response + par_CMN_Transmission_Delay);

       alt{
           []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_SLAC_MATCH_REQ := '607C'H}),
                              md_CMN_CMN_CmSlacMatchReq_001(
                              m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                              par_testSystem_mac, vc_RunID))) {

               v_count := v_count + 1;
               tc_TT_match_response.start(par_TT_match_response + par_CMN_Transmission_Delay);

               if(v_count > par_C_EV_match_retry) {
                 alt{
                   [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                     md_CMN_CMN_SlacMmeCmnHeader_001({
                                     CM_SLAC_MATCH_REQ := '607C'H}),
                                     md_CMN_CMN_CmSlacMatchReq_001(
                                     m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                                     par_testSystem_mac, vc_RunID))) {

                      setverdict(fail,"CM_SLAC_MATCH.REQ message was repeated, but v_count > " &
                                   "par_C_EV_match_retry.");
                   }
                   [] tc_TT_match_response.timeout {
                      setverdict(pass,"TT_match_response timeout. " &
                                   "The total number of retries is reached, " &
                                   "the Matching process " &
                                   "shall be considered as FAILED.");
                    }
                 }
               }
               else{
                   repeat;
               }
            }
            [] pt_SLAC_Port.receive {
               setverdict(fail, "Invalid message type or content was received.");
            }
           [] tc_TT_EVSE_match_session.timeout {
              setverdict(fail,"TT_EVSE_match_session timeout. Matching process shall be " &
                              "considered as FAILED.");
           }
           [] tc_TT_match_response.timeout {
              setverdict(fail,"TT_match_response timeout. " &
                              "CM_SLAC_MATCH.REQ message was not repeated.");
           }
       }
       return getverdict;
   }

   function f_EVCC_CMN_TB_VTB_CmSlacMatch_003(in template(present) MME_Payload mmePayload)
                                              runs on EVCC_Tester return verdicttype {

       var MME v_responseMessage;
       var MACAddress_TYPE v_address;
       var integer v_count := 0;
       var boolean v_stopLnkStatus := false;

       v_count := v_count + 1;
       tc_TT_match_response.start(par_TT_match_response + par_CMN_Transmission_Delay);

       // send invalid CM_SLAC_MATCH.CNF message
       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                         md_CMN_CMN_SlacMmeCmnHeader_001({
                         CM_SLAC_MATCH_CNF := '607D'H}),
                         mmePayload)) to vc_sut_mac;

       alt{
           []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                  md_CMN_CMN_SlacMmeCmnHeader_001({
                                  CM_SLAC_MATCH_REQ := '607C'H}),
                                  md_CMN_CMN_CmSlacMatchReq_001(
                                  m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                                  par_testSystem_mac, vc_RunID))) {

               v_count := v_count + 1;
               tc_TT_match_response.start(par_TT_match_response + par_CMN_Transmission_Delay);

               // send invalid CM_SLAC_MATCH.CNF message
               pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_SLAC_MATCH_CNF := '607D'H}),
                                 mmePayload)) to vc_sut_mac;

               if(v_count > par_C_EV_match_retry) {
                 alt{
                   []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_SLAC_MATCH_REQ := '607C'H}),
                                          md_CMN_CMN_CmSlacMatchReq_001(
                                          m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                                          par_testSystem_mac, vc_RunID))) {

                     setverdict(fail,"CM_SLAC_MATCH.REQ message was repeated, but v_count > " &
                                     "par_C_EV_match_retry.");
                   }
                   [] tc_TT_match_response.timeout {
                     setverdict(pass,"TT_match_response timeout. " &
                                     "The total number of retries is reached, " &
                                     "the Matching process " &
                                     "shall be considered as FAILED");
                   }
                 }
               }
               else{
                   repeat;
               }
           }
           [] pt_SLAC_Port.receive {
              setverdict(fail, "Invalid message type or content was received.");
           }
           [] tc_TT_EVSE_match_session.timeout {
              setverdict(fail,"TT_EVSE_match_session timeout. Matching process shall be " &
                              "considered as FAILED.");
           }
           [] tc_TT_match_response.timeout {
              setverdict(fail,"TT_match_response timeout. " &
                              "CM_SLAC_MATCH.REQ message was not repeated.");
            }
        }
        return getverdict;
    }
}
```

E.2.6 EVCC functions for CmSetKey

```
module TestBehavior_EVCC_CmSetKey {

    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Templates_CMN_CmSetKey all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from Services_PLCLinkStatus all;
    import from Timer_15118 all;

    function f_EVCC_CMN_TB_VTB_CmSetKey_001() runs on EVCC_Tester return verdicttype {

        timer t1 := par_CMN_setKey;
        t1.start;

        pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                          md_CMN_CMN_SlacMmeCmnHeader_001({
                          CM_SET_KEY_REQ := '6008'H}),
                          md_CMN_CMN_CmSetKeyReq_001(vc_Nid, vc_Nmk)))
                          to par_testSystem_plc_node_mac;

         alt {
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SET_KEY_CNF := '6009'H}),
                                       mdw_CMN_CMN_CmSetKeyCnf_001('01'H))) {

                    setverdict(pass,"CM_SET_KEY is correct.");
                }
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_SET_KEY_CNF := '6009'H}),
                                       mdw_CMN_CMN_CmSetKeyCnf_001('00'H))) {

                    setverdict(inconc,"CM_SET_KEY is incorrect. " &
                                     "The PLC node could not set the key.");
                }
                [] a_EVCC_processPLCLinkNotifications_001();
                [] pt_SLAC_Port.receive {
                   setverdict(inconc, "Invalid message type or content was received.");
                }
                [] t1.timeout {
                   setverdict(inconc,"CM_SET_KEY timeout.");
                }
        }
        return getverdict;
      }
}
```

E.2.7 EVCC functions for PLCLinkStatus

```
module TestBehavior_EVCC_PLCLinkStatus {

    import from Timer_15118_3 all;
    import from Pics_15118 all;
    import from Pixit_15118_3 all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmSlacMatch all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from Services_PLCLinkStatus all;
    import from Services_HAL_61851 all;
    import from DataStructure_HAL_61851 all;
    import from TestBehavior_EVCC_SDP all;
    import from LibFunctions_15118_3 all;
import from Timer_15118 all;
import from Pixit_15118 all;
import from TestBehavior_EVCC_CmSetKey all;
import from TestBehavior_EVCC_CmSlacParm all;
import from Templates_CMN_CmNwStats all;

function f_EVCC_CMN_TB_VTB_PLCLinkStatus_001(in verdicttype v_vct)
                                             runs on EVCC_Tester
                                             return verdicttype {

     var verdicttype v_verdict;

    tc_TT_match_join.start(par_TT_match_join);

    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_SLAC_MATCH_CNF := '607D'H}),
                      md_CMN_CMN_CmSlacMatchCnf_001(
                      m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                      par_testSystem_mac, vc_RunID, vc_Nid,vc_Nmk)))
                      to vc_sut_mac;

    v_verdict := f_EVCC_getPLCLinkEstablishment(v_vct);

    if(v_verdict == pass) {
        setverdict(pass, "The data link was established by the SUT.");
    }
    else {
        setverdict(v_vct, "The data link could not be established by the SUT.");
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_PLCLinkStatus_002() runs on EVCC_Tester return verdicttype {

    var verdicttype v_verdict;

    // set state E
        f_EVCC_changeValidStateCondition(E,E);
        f_EVCC_setPwmMode(e_OscOff);

    v_verdict := f_EVCC_getPLCLinkTermination(par_TP_match_leave +
                                              par_CMN_Transmission_Delay, fail);

    if(v_verdict == pass) {
        setverdict(pass, "The data link was terminated by the SUT.");
    }
    else {
        setverdict(fail, "The data link did not terminated by the SUT.");
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_PLCLinkStatus_003(HAL_61851_PwmMode_Type pwmMode)
                                             runs on EVCC_Tester
                                             return verdicttype {
    var verdicttype v_verdict;

    // set error state
        f_EVCC_changeValidStateCondition(E,E);
        f_EVCC_setPwmMode(pwmMode);

    tc_TT_match_join.start(par_TT_match_join);

    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                      md_CMN_CMN_SlacMmeCmnHeader_001({
                      CM_SLAC_MATCH_CNF := '607D'H}),
                      md_CMN_CMN_CmSlacMatchCnf_001(
                      m_CMN_CMN_SlacPayloadHeader_001(), vc_sut_mac,
                      par_testSystem_mac, vc_RunID,vc_Nid,vc_Nmk)))
                      to vc_sut_mac;

    v_verdict := f_EVCC_getPLCLinkError();

    if(v_verdict == pass) {
        setverdict(pass, "The data link was not established by the SUT.");
    }
    else {
        setverdict(fail, "The data link was established by the SUT.");
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_PLCLinkStatus_004(in HAL_61851_Listener
                                             v_HAL_61851_Listener)
                                             runs on EVCC_Tester
                                             return verdicttype {

  var verdicttype verdict := pass;

  v_HAL_61851_Listener.stop;
  v_HAL_61851_Listener.start(f_EVCC_HAL61851Listener(true));

  f_EVCC_setPwmMode(e_OscOn);
  f_EVCC_setDutyCycle(5);

  f_EVCC_getPLCLinkEstablishmentAfterSleepMode(fail);

  if(getverdict == pass) {
      tc_V2G_SECC_CommunicationSetup_Timer.start;
      verdict := f_EVCC_CMN_TB_VTB_SDP_001(?, fail);
  }
  return getverdict;
}

function f_EVCC_CMN_TB_VTB_PLCLinkStatus_005(in HAL_61851_Listener
                                             v_HAL_61851_Listener,
                                             in float v_time)
                                             runs on EVCC_Tester
                                             return verdicttype {

  var verdicttype verdict := pass;

  v_HAL_61851_Listener.stop;
  v_HAL_61851_Listener.start(f_EVCC_HAL61851Listener(true));

  // BCB toggle sequence detection
  f_EVCC_changeValidStateCondition(B,C);
  timer statetimer := (PICS_CMN_CMN_WakeUp -
                       v_time + 5.0);
  statetimer.start;

  alt {
     [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, B){

         statetimer.stop;
     }
     [] a_EVCC_BCBToggleDetection(pt_HAL_61851_Internal_Port, C){

         statetimer.stop;
         f_EVCC_changeValidStateCondition(C,B);
         statetimer.start(par_T_vald_state_duration_max);
         repeat;
     }
     [] pt_HAL_61851_Internal_Port.receive {
         setverdict(fail, "Received state has an invalid value.");
     }
     [] statetimer.timeout {
         setverdict(fail, "The EVSE could not detect the corresponding " &
                          "toggle value within " &
                          "the maximal valid state duration.");
     }
  }

  if(getverdict != pass) {
     log("The SUT did not initiate a wake-up " &
         "within 'PICS_CMN_CMN_WakeUp'.");
  }

  f_EVCC_getPLCLinkEstablishmentAfterSleepMode(fail);

  if(getverdict == pass) {
      tc_V2G_SECC_CommunicationSetup_Timer.start;
      verdict := f_EVCC_CMN_TB_VTB_SDP_001(?, fail);
  }

  return getverdict;
```

```
    }

    function f_EVCC_CMN_TB_VTB_PLCLinkStatus_006(in HAL_61851_Listener
                                             v_HAL_61851_Listener,
                                             in integer v_dutyCycle,
                                             in IEC_61851_States v_state)
                                             runs on EVCC_Tester
                                             return verdicttype {

        var verdicttype v_verdict := pass;

        sleep(par_CMN_waitForConnectionLoss);

        // generate new Nid and Nmk
        vc_Nmk := f_randomHexStringGen(32);
        vc_Nid := fx_generateNID(vc_Nmk);
        v_verdict := f_EVCC_CMN_TB_VTB_CmSetKey_001();
        if (v_verdict == pass) {
            v_verdict := f_EVCC_getPLCLinkTermination(par_TP_match_leave, fail);
        }

        if (v_verdict == pass) {
            v_verdict := f_EVCC_setPwmMode(e_PosVolt12);
        }

        sleep(par_CMN_waitForNextHAL);

        if (v_verdict == pass) {
            f_EVCC_changeValidStateCondition(E,E);
            v_verdict := f_EVCC_setPwmMode(e_OscOff);
        }

        tc_T_step_EF.start(par_T_step_EF_min);
        alt {
            [] tc_T_step_EF.timeout {}
            [] a_EVCC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or " &
                                 "content was received.");
            }
        }

        if (v_verdict == pass) {
            f_EVCC_changeValidStateCondition(v_state,valid);
            v_verdict := f_EVCC_setDutyCycle(v_dutyCycle);
        }

        sleep(par_CMN_waitForNextHAL);

        if (v_verdict == pass) {
            v_verdict := f_EVCC_setPwmMode(e_OscOn);
        }

        if (v_verdict == pass) {
            timer statetimer := par_CMN_HAL_Timeout;
            v_verdict := f_EVCC_confirmState(valid, v_HAL_61851_Listener,
                                            statetimer, fail);
        }

        if (v_verdict == pass) {
            tc_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min);
            f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        }
        return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_PLCLinkStatus_007() runs on EVCC_Tester
                                                   return verdicttype {

        tc_TP_match_leave.start(par_TP_match_leave);
        alt {
            [] tc_TP_match_leave.timeout {}
            [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or " &
                                 "content was received.");
            }
        }

        tc_TT_link_status_response.start;
    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(md_CMN_CMN_SlacMmeCmnHeader_001({
                                             CM_NW_STATS_REQ := '6048'H}),
                                             md_CMN_CMN_CmNwStatsReq_001()))
                                             to par_testSystem_plc_node_mac;

    alt {
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_NW_STATS_CNF := '6049'H}),
                                md_CMN_CMN_CmNwStatsCnf_001())) {

           setverdict(fail,"The SUTs node was detected in the current " &
                           "logical network.");
        }
        [] pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                md_CMN_CMN_SlacMmeCmnHeader_001({
                                CM_NW_STATS_CNF := '6049'H}),
                                md_CMN_CMN_CmNwStatsCnf_002())) {

           setverdict(pass,"The SUTs node has left the current " &
                           "logical network.");
        }
        [] pt_SLAC_Port.receive {
            setverdict(fail, "Invalid message type or " &
                             "content was received.");
        }
        [] tc_TT_link_status_response.timeout {
            setverdict(fail,"CM_NW_STATS timeout.");
        }
    }
    return getverdict;
}

function f_EVCC_CMN_TB_VTB_PLCLinkStatus_008(in HAL_61851_Listener
                                             v_HAL_61851_Listener)
                                             runs on EVCC_Tester
                                              return verdicttype {

    var verdicttype v_verdict := pass;
    var hexstring v_Nmk_old;
    var hexstring v_Nid_old;

    v_Nmk_old := vc_Nmk;
    v_Nid_old := vc_Nid;

    sleep(par_CMN_waitForConnectionLoss);

    v_HAL_61851_Listener.stop;
    v_HAL_61851_Listener.start(f_EVCC_HAL61851Listener(false));

    // generate new Nid and Nmk
    vc_Nmk := f_randomHexStringGen(32);
    vc_Nid := fx_generateNID(vc_Nmk);
    v_verdict := f_EVCC_CMN_TB_VTB_CmSetKey_001();
    if (v_verdict == pass) {
        v_verdict := f_EVCC_getPLCLinkTermination(par_TP_match_leave, fail);
    }

    // set old Nid and Nmk
    if (v_verdict == pass) {
        vc_Nmk := vc_Nmk;
        vc_Nid := vc_Nid;
        v_verdict := f_EVCC_CMN_TB_VTB_CmSetKey_001();
    }

    if (v_verdict == pass) {
        v_verdict := f_EVCC_checkLeavingLogicalNetwork();
    }
    return getverdict;
}

function f_EVCC_AC_TB_VTB_PLCLinkStatus_001() runs on EVCC_Tester
                                              return verdicttype {

    var verdicttype v_verdict := pass;

    sleep(par_CMN_waitForConnectionLoss);
```

```
        // generate new Nid and Nmk
        vc_Nmk := f_randomHexStringGen(32);
        vc_Nid := fx_generateNID(vc_Nmk);
        v_verdict := f_EVCC_CMN_TB_VTB_CmSetKey_001();
        if (v_verdict == pass) {
            v_verdict := f_EVCC_getPLCLinkTermination(par_TP_match_leave, fail);
        }

        tc_TconnResetup.start(PIXIT_EVCC_AC_TconnResetup);
        alt {
            [] tc_TconnResetup.timeout {}
            [] a_EVCC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content " &
                                 "was received.");
            }
        }

        if (v_verdict == pass) {
            tc_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min);
            v_verdict := f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        }
        return getverdict;
    }

    function f_EVCC_AC_TB_VTB_PLCLinkStatus_002(in HAL_61851_Listener
                                             v_HAL_61851_Listener,
                                             in IEC_61851_States v_state)
                                             runs on EVCC_Tester
                                             return verdicttype {

        var verdicttype v_verdict := pass;

        sleep(par_CMN_waitForConnectionLoss);

        // generate new Nid and Nmk
        vc_Nmk := f_randomHexStringGen(32);
        vc_Nid := fx_generateNID(vc_Nmk);
        v_verdict := f_EVCC_CMN_TB_VTB_CmSetKey_001();
        if (v_verdict == pass) {
            v_verdict := f_EVCC_getPLCLinkTermination(par_TP_match_leave, fail);
        }

        tc_TconnResetup.start(PIXIT_EVCC_AC_TconnResetup/2.0);
        alt {
            [] tc_TconnResetup.timeout {}
            [] a_EVCC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content " &
                                 "was received.");
            }
        }

        if (v_verdict == pass) {
            v_verdict := f_EVCC_setPwmMode(e_PosVolt12);
        }

        sleep(par_CMN_waitForNextHAL);

        if (v_verdict == pass) {
            f_EVCC_changeValidStateCondition(E,E);
            v_verdict := f_EVCC_setPwmMode(e_OscOff);
        }

        tc_T_step_EF.start(par_T_step_EF_min);
        alt {
            [] tc_T_step_EF.timeout {}
            [] a_EVCC_processPLCLinkNotifications_001();
            [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or " &
                                 "content was received.");
            }
        }

        if (v_verdict == pass) {
            f_EVCC_changeValidStateCondition(v_state, valid);
            v_verdict := f_EVCC_setDutyCycle(par_EVSENominalDutyCycle);
        }
        sleep(par_CMN_waitForNextHAL);

        if (v_verdict == pass) {
            v_verdict := f_EVCC_setPwmMode(e_OscOn);
        }

        if (v_verdict == pass) {
            timer statetimer := par_CMN_HAL_Timeout;
            v_verdict := f_EVCC_confirmState(valid, v_HAL_61851_Listener,
                                             statetimer, fail);
        }

        if (v_verdict == pass) {
            tc_TT_EVSE_SLAC_init.start(par_TT_EVSE_SLAC_init_min);
            f_EVCC_CMN_TB_VTB_CmSlacParm_001(fail);
        }
        return getverdict;
    }
}
```

E.2.8 EVCC functions for CmAmpMap

```
module TestBehavior_EVCC_CmAmpMap {

    import from Timer_15118_3 all;
    import from Pics_15118_3 all;
    import from Templates_CMN_CmAmpMap all;
    import from Templates_CMN_SlacManagementMessageEntry all;
    import from Templates_CMN_SlacPayloadHeader all;
    import from Templates_CMN_CmSlacMatch all;
    import from ComponentsAndPorts all;
    import from DataStructure_SLAC all;
    import from Services_HAL_61851 all;
    import from DataStructure_HAL_61851 all;
    import from Templates_CMN_CmSlacParm all;
    import from TTlibrary_Logging all;
    import from TestBehavior_EVCC_SDP all;

    function f_EVCC_CMN_TB_VTB_CmAmpMap_001(in verdicttype v_vct)
                                        runs on EVCC_Tester
                                        return verdicttype {

       var boolean v_repetition := true;
       var integer v_counter := 0;

       while(v_repetition){

           tc_TT_match_response.start(par_TT_match_response);
           v_counter := v_counter + 1;
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_AMP_MAP_REQ := '601C'H}),
                              m_CMN_CMN_CmAmpMapReq_001()))
                              to vc_sut_mac;

            alt {
                   []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                              md_CMN_CMN_SlacMmeCmnHeader_001({
                                              CM_AMP_MAP_CNF := '601D'H}),
                                           md_CMN_CMN_CmAmpMapCnf_001('00'H))) {

                       setverdict(pass,"CM_AMP_MAP.CNF is correct.");
                       v_repetition := false;
                       tc_TT_match_response.stop;
                   }

                   []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                              md_CMN_CMN_SlacMmeCmnHeader_001({
                                              CM_AMP_MAP_CNF := '601D'H}),
                                           md_CMN_CMN_CmAmpMapCnf_001('01'H))) {

                       setverdict(v_vct,"The SUT could not perform the " &
                                        "Amplitude map exchange.");
                       v_repetition := false;
                       tc_TT_match_response.stop;
                   }
```

```
                [] pt_SLAC_Port.receive {

                   setverdict(v_vct, "Invalid message type or content " &
                                     "was received.");
                   v_repetition := false;
                   tc_TT_match_response.stop;
                }
                [] tc_TT_match_response.timeout {
                    log("TT_match_response timeout.");
                    if(v_counter mod (par_C_EV_match_retry+1) == 0){
                       setverdict(v_vct,"The SUT did not response to the " &
                                        "CM_AMP_MAP.REQ message.");
                       v_repetition := false;
                    } else {
                       log("A new CM_AMP_MAP.REQ message will be sent.");
                    }
                }
            }
        }
        return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmAmpMap_002(in verdicttype v_vct)
                                            runs on EVCC_Tester
                                            return verdicttype {

        var MME v_requestMessage;

         tc_TT_amp_map_exchange.start(par_TT_amp_map_exchange);

         alt {
                []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                       md_CMN_CMN_SlacMmeCmnHeader_001({
                                       CM_AMP_MAP_REQ := '601C'H}),
                                       md_CMN_CMN_CmAmpMapReq_002(?,?)))
                                       -> value v_requestMessage {

                    pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_AMP_MAP_CNF := '601D'H}),
                                      md_CMN_CMN_CmAmpMapCnf_001('00'H)))
                                      to vc_sut_mac;

                    var Amlen_TYPE v_amlen := v_requestMessage.mme_payload.
                                              payload.cm_amp_map_req.amlen;
                    var ListofAmdata_TYPE v_listAmdata := v_requestMessage.mme_payload.
                                                          payload.cm_amp_map_req.listAmdata;

                    setverdict(pass,"CM_AMP_MAP.REQ is correct.");
                    tc_TT_amp_map_exchange.stop;
                }
                [] pt_SLAC_Port.receive {
                   setverdict(v_vct, "Invalid message type or content was received.");
                }
                [] tc_TT_amp_map_exchange.timeout {
                    setverdict(v_vct,"No Amplitude Map exchange was performed by the SUT.");
                }
          }
          return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmAmpMap_003() runs on EVCC_Tester return verdicttype {

         var integer v_count := 0;
         tc_TT_amp_map_exchange.start(par_TT_amp_map_exchange);

         alt {
             []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                    md_CMN_CMN_SlacMmeCmnHeader_001({
                                    CM_AMP_MAP_REQ := '601C'H}),
                                    md_CMN_CMN_CmAmpMapReq_002(?,?))) {

                if(v_count > 0){
                    setverdict(pass,"CM_AMP_MAP.REQ message was repeated.",v_count);
                } else { tc_TT_amp_map_exchange.stop;}

                v_count := v_count + 1;
                tc_TT_match_response.start(par_TT_match_response);
            if(v_count > par_C_EV_match_retry) {

                alt{
                    []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                          md_CMN_CMN_SlacMmeCmnHeader_001({
                                          CM_AMP_MAP_REQ := '601C'H}),
                                          md_CMN_CMN_CmAmpMapReq_002(?,?))) {

                      setverdict(fail,"CM_AMP_MAP.REQ message was repeated, " &
                                       "but v_count > par_C_EV_match_retry.");
                    }
                    [] pt_SLAC_Port.receive {
                       setverdict(fail, "Invalid message type or content " &
                                          "was received.");
                    }
                    [] tc_TT_match_response.timeout {
                      setverdict(pass,"TT_match_response timeout. " &
                                     "The total number of retries is reached, " &
                                     "the Matching process " &
                                     "shall be considered as FAILED.");
                    }
                  }
            }
            else{
                repeat;
            }
        }
        [] pt_SLAC_Port.receive {
           setverdict(fail, "Invalid message type or content was received.");
        }
        [] tc_TT_amp_map_exchange.timeout {
            setverdict(fail,"No Amplitude Map exchange was performed by the SUT.");
        }
        [] tc_TT_match_response.timeout {
            setverdict(fail,"The SUT did not retransmit the " &
                            "CM_AMP_MAP.REQ message.");
        }
     }
     return getverdict;
 }

function f_EVCC_CMN_TB_VTB_CmAmpMap_004() runs on EVCC_Tester return verdicttype {

     var integer v_count := 0;
     tc_TT_amp_map_exchange.start(par_TT_amp_map_exchange);

     alt {
         []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                               md_CMN_CMN_SlacMmeCmnHeader_001({
                               CM_AMP_MAP_REQ := '601C'H}),
                               md_CMN_CMN_CmAmpMapReq_002(?,?))) {

             if(v_count > 0){
                 setverdict(pass,"CM_AMP_MAP.REQ message was repeated.",v_count);
             } else { tc_TT_amp_map_exchange.stop;}

             v_count := v_count + 1;
             tc_TT_match_response.start(par_TT_match_response);
             // send invalid CM_AMP_MAP.CNF message
             pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_AMP_MAP_CNF := '601D'H}),
                              md_CMN_CMN_CmAmpMapCnf_001('FF'H))
                              to vc_sut_mac;

             if(v_count > par_C_EV_match_retry) {

                 alt{
                     []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                           md_CMN_CMN_SlacMmeCmnHeader_001({
                                           CM_AMP_MAP_REQ := '601C'H}),
                                           md_CMN_CMN_CmAmpMapReq_002(?,?))) {

                       setverdict(fail,"CM_AMP_MAP.REQ message was repeated, " &
                                      "but v_count > par_C_EV_match_retry.");
                     }
                     [] pt_SLAC_Port.receive {
                        setverdict(fail, "Invalid message type or content " &
                                        "was received.");
                        }
                        [] tc_TT_match_response.timeout {
                          setverdict(pass,"TT_match_response timeout. " &
                                          "The total number of retries is reached, " &
                                          "the Matching process " &
                                          "shall be considered as FAILED.");
                        }
                      }
                 }
                 else{
                     repeat;
                 }
             }
             [] pt_SLAC_Port.receive {
                setverdict(fail, "Invalid message type or content was received.");
             }
             [] tc_TT_amp_map_exchange.timeout {
                 setverdict(fail,"No Amplitude Map exchange was performed by the SUT.");
             }
             [] tc_TT_match_response.timeout {
                 setverdict(fail,"The SUT did not retransmit the " &
                                 "CM_AMP_MAP.REQ message.");
             }
         }
         return getverdict;
    }

    function f_EVCC_CMN_TB_VTB_CmAmpMap_005() runs on EVCC_Tester return verdicttype {

       var boolean v_repetition := true;
       var integer v_counter := 0;

       while(v_repetition){

           tc_TT_match_response.start(par_TT_match_response);
           v_counter := v_counter + 1;
           // send invalid CM_AMP_MAP.REQ message
           pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                             md_CMN_CMN_SlacMmeCmnHeader_001({
                             CM_AMP_MAP_REQ := '601C'H}),
                             md_CMN_CMN_CmAmpMapReq_003('0000'H)))
                             to vc_sut_mac;

           alt {
               []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_AMP_MAP_CNF := '601D'H}),
                                      md_CMN_CMN_CmAmpMapCnf_001(?))) {

                   setverdict(fail,"Received CM_AMP_MAP.CNF message " &
                                   "was not expected.");
                   v_repetition := false;
                   tc_TT_match_response.stop;
               }
               [] pt_SLAC_Port.receive {

                  setverdict(fail, "Invalid message type or content " &
                                   "was received.");
                  v_repetition := false;
                  tc_TT_match_response.stop;
               }
               [] tc_TT_match_response.timeout {
                   log("TT_match_response timeout.");
                   setverdict(pass,"The SUT did not response to the " &
                                   "invalid CM_AMP_MAP.REQ message.");
                   if(v_counter mod (par_C_EV_match_retry+1) == 0){
                      v_repetition := false;
                   } else {
                      log("A new invalid CM_AMP_MAP.REQ message will be sent.");
                   }
               }
            }
       }
       return getverdict;
   }

   function f_EVCC_CMN_TB_VTB_CmAmpMap_006() runs on EVCC_Tester return verdicttype {
```

```
   var boolean v_repetition := true;
   var integer v_counter := 0;

   while(v_repetition){

       tc_TT_match_response.start(par_TT_match_response);
       v_counter := v_counter + 1;
       pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                         md_CMN_CMN_SlacMmeCmnHeader_001({
                         CM_AMP_MAP_REQ := '601C'H}),
                         m_CMN_CMN_CmAmpMapReq_001()))
                         to vc_sut_mac;

        alt {
               []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_AMP_MAP_CNF := '601D'H}),
                                      md_CMN_CMN_CmAmpMapCnf_001('00'H))) {

                   setverdict(pass,"CM_AMP_MAP.CNF is correct.");
                   tc_TT_match_response.stop;
                   if(v_counter > 1) {
                       v_repetition := false;
                   }
               }
               []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                      md_CMN_CMN_SlacMmeCmnHeader_001({
                                      CM_AMP_MAP_CNF := '601D'H}),
                                      md_CMN_CMN_CmAmpMapCnf_001('01'H))) {

                   setverdict(fail,"The SUT could not perform the " &
                                   "Amplitude map exchange.");
                   v_repetition := false;
                   tc_TT_match_response.stop;
               }
               [] pt_SLAC_Port.receive {

                  setverdict(fail, "Invalid message type or content " &
                                   "was received.");
                  v_repetition := false;
                  tc_TT_match_response.stop;
               }
               [] tc_TT_match_response.timeout {
                  setverdict(fail,"The SUT did not response to the " &
                                 "CM_AMP_MAP.REQ message.");
                  v_repetition := false;
               }
        }
   }
   return getverdict;
}

function f_EVCC_CMN_TB_VTB_CmAmpMap_007() runs on EVCC_Tester return verdicttype {

   var integer v_counter := 0;

   tc_TT_match_response.start(par_TT_match_response);
   for (var integer i:=0; i<3; i:=i + 1) {
     pt_SLAC_Port.send(md_CMN_CMN_SlacMme_001(
                       md_CMN_CMN_SlacMmeCmnHeader_001({
                       CM_AMP_MAP_REQ := '601C'H}),
                       m_CMN_CMN_CmAmpMapReq_001()))
                       to vc_sut_mac;
   }

   alt {
       []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                              md_CMN_CMN_SlacMmeCmnHeader_001({
                              CM_AMP_MAP_CNF := '601D'H}),
                              md_CMN_CMN_CmAmpMapCnf_001('00'H))) {

           setverdict(pass,"CM_AMP_MAP.CNF is correct.");
           v_counter := v_counter + 1;
           tc_TT_match_response.stop;
           tc_TT_match_response.start(par_TT_match_response);
           if(v_counter < 3) {
               repeat;
           }
```

```
          }
          []pt_SLAC_Port.receive(md_CMN_CMN_SlacMme_001(
                                 md_CMN_CMN_SlacMmeCmnHeader_001({
                                 CM_AMP_MAP_CNF := '601D'H}),
                                 md_CMN_CMN_CmAmpMapCnf_001('01'H))) {

              setverdict(fail,"The SUT could not perform the " &
                              "Amplitude map exchange.");
              tc_TT_match_response.stop;
          }
          [] pt_SLAC_Port.receive {

             setverdict(fail, "Invalid message type or content " &
                              "was received.");
             tc_TT_match_response.stop;
          }
          [] tc_TT_match_response.timeout {
             setverdict(fail,"The SUT did not response to the " &
                             "CM_AMP_MAP.REQ message.");
          }
       }
      return getverdict;
   }

   function f_EVCC_CMN_TB_VTB_CmAmpMap_008() runs on EVCC_Tester return verdicttype {

      tc_V2G_SECC_CommunicationSetup_Timer.start;
      f_EVCC_CMN_TB_VTB_SDP_001(?, fail);

      return getverdict;
   }
}
```

Annex F
(normative)

Template specifications for 15118-3

F.1 Common + PLC bridge templates

This subclause includes all template *specifications* which are defined for EV and EVSE.

```
module Templates_CMN_SlacPayloadHeader {

  import from DataStructure_SLAC  all;

  template SLAC_Header m_CMN_CMN_SlacPayloadHeader_001() := {
     application_type := '00'H,
     security_type := '00'H
  }

  template SLAC_Header m_CMN_CMN_SlacPayloadHeaderInvalid_001() := {
     application_type := 'FF'H,
     security_type := '00'H
  }

  template SLAC_Header m_CMN_CMN_SlacPayloadHeaderInvalid_002() := {
     application_type := '00'H,
     security_type := 'FF'H
  }
}


module Templates_CMN_SlacManagementMessageEntry {

    import from DataStructure_SLAC  all;

    template MME md_CMN_CMN_SlacMme_001(
        template(present) MME_Header p_mme_header,
        template(present) MME_Payload p_mme_payload) := {

        mme_header := p_mme_header,
        mme_payload := p_mme_payload
    }

    template MME_Header md_CMN_CMN_SlacMmeCmnHeader_001(
        template(present) MMTYPE p_mmtype) := {

         mmv := '01'H,
         mmtype := p_mmtype,
         fmi := '00'H,
         fmsn := '00'H
    }

    template MME_Header md_CMN_CMN_SlacMmeOuiHeader_001(
        template(present) MMTYPE p_mmtype) := {

        mmv := '00'H,
        mmtype := p_mmtype,
        fmi := omit,
        fmsn := omit
    }
}
```

F.1.1 CMN templates for CmSlacParm

```
module Templates_CMN_CmSlacParm {

   import from DataStructure_SLAC  all;

   template MME_Payload md_CMN_CMN_CmSlacParmReq_001 (
        template(present) SLAC_Header v_slac_header,
        template(present) RunID_TYPE  v_runid) := {

       payload := {
           cm_slac_parm_req := {
                 slac_header := v_slac_header,
                 runid := v_runid
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmSlacParmCnf_001 (
      template(present) MACAddress_TYPE p_forwarding_sta,
      template(present) SLAC_Header p_appheader,
      template(present) RunID_TYPE p_runid) := {

           payload := {
               cm_slac_parm_cnf := {
                        msound_target := 'FFFFFFFFFFFF'H,
                        num_sounds := '0A'H,
                        time_out := '06'H,
                        resp_type := '01'H,
                        forwarding_sta := p_forwarding_sta,
                        appheader := p_appheader,
                        runid := p_runid
                     }
           }
      }

    template MME_Payload md_CMN_CMN_CmSlacParmCnf_002 (

       template(present) MACAddress_TYPE p_msound_target,
       template(present) NumSounds_TYPE p_num_sounds,
       template(present) TimeOut_TYPE p_time_out,
       template(present) RespType_TYPE p_resp_type,
       template(present) MACAddress_TYPE p_forwarding_sta,
       template(present) SLAC_Header p_appheader,
       template(present) RunID_TYPE p_runid) := {

            payload := {
                cm_slac_parm_cnf := {
                         msound_target := p_msound_target,
                         num_sounds := p_num_sounds,
                         time_out := p_time_out,
                         resp_type := p_resp_type,
                         forwarding_sta := p_forwarding_sta,
                         appheader := p_appheader,
                         runid := p_runid
                      }
            }
     }

     template MACAddressList_TYPE m_CMN_CMN_EmptyMacAddresList() := {
        macAddressList := omit
     }
}
```

F.1.2 CMN templates for CmStartAttenCharInd

```
module Templates_CMN_CmStartAttenCharInd {

    import from DataStructure_SLAC  all;

    template MME_Payload md_CMN_CMN_CmStartAttenCharInd_001(
        template(present) SLAC_Header v_slac_header,
        template(present) NumSounds_TYPE v_num_sounds,
        template(present) TimeOut_TYPE v_time_out,
        template(present) RespType_TYPE v_resp_type,
        template(present) MACAddress_TYPE v_forwarding_sta,
        template(present) RunID_TYPE v_runid) := {

        payload:= {
           cm_start_atten_char_ind := {
               slac_header := v_slac_header,
               num_sounds := v_num_sounds,
               time_out := v_time_out,
               resp_type := v_resp_type,
               forwarding_sta := v_forwarding_sta,
               runid := v_runid
                  }
           }
       }
}
```

F.1.3 CMN templates for CmMnbcSoundInd

```
module Templates_CMN_CmMnbcSoundInd {

  import from DataStructure_SLAC  all;

  template MME_Payload md_CMN_CMN_CmMnbcSoundInd_001(
          template(present) SLAC_Header   v_slac_header,
        template(present) Count_TYPE   v_count,
        template(present) RunID_TYPE   v_runid,
        template(present) SourceRnd_Type   v_source_rnd):= {

        payload:= {
           cm_mnbc_sound_ind := {
               slac_header := v_slac_header,
               source_id := '0000000000000000000000000000000000'H,
               count := v_count,
               runid := v_runid,
               res0 := '0000000000000000'H,
               source_rnd := v_source_rnd }
                  }
       }
}
```

F.1.4 CMN templates for CmAttenCharRsp

```
module Templates_CMN_CmAttenCharRsp {

  import from DataStructure_SLAC  all;

  template Acvarfield_Type md_CMN_CMN_Acvarfield_001 (
      template(present) MACAddress_TYPE   v_source_address,
      template(present) RunID_TYPE   v_runid):= {

      source_address := v_source_address,
      runid := v_runid,
      source_id := '0000000000000000000000000000000000'H,
      resp_id := '0000000000000000000000000000000000'H,
      result := '00'H
  }

  template Acvarfield_Type md_CMN_CMN_Acvarfield_002 (
      template(present) MACAddress_TYPE   p_source_address,
      template(present) RunID_TYPE   p_runid,
      template(present) StationID_TYPE p_source_id,
      template(present) StationID_TYPE p_resp_id,
      template(present) Result_TYPE p_result):= {
```

```
      source_address := p_source_address,
      runid := p_runid,
      source_id := p_source_id,
      resp_id := p_resp_id,
      result := p_result
  }

  template MME_Payload md_CMN_CMN_CmAttenCharRsp_001(
      template(present) SLAC_Header   v_slac_header,
      template(present) Acvarfield_Type v_acvarfield):= {

      payload:= {
          cm_atten_char_rsp := {
             slac_header := v_slac_header,
             acvarfield := v_acvarfield}
                }
     }
}
```

F.1.5 CMN templates for CmValidate

```
module Templates_CMN_CmValidate {

    import from DataStructure_SLAC  all;

    template MME_Payload m_CMN_CMN_CmValidateReq_001() := {
        {
            cm_validate_req := {
                signalType := '00'H,
                vrVarField := {
                    pilot_timer := '00'H,
                    result := '01'H
                }
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmValidateReq_002(
        in template(present) PilotTimer_TYPE p_pilot_timer) := {

        {
            cm_validate_req := {
                signalType := '00'H,
                vrVarField := {
                    pilot_timer := p_pilot_timer,
                    result := '01'H
                }
            }
        }
    }

    template MME_Payload mw_CMN_CMN_CmValidateReq_003() := {
        {
            cm_validate_req := {
                signalType := '00'H,
                vrVarField := {
                    pilot_timer := ?,
                    result := ?
                }
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmValidateReq_004(
        in template(present) SignalType_TYPE p_signalType,
        in template(present) PilotTimer_TYPE p_pilot_timer,
        in template(present) Result_TYPE p_result) := {

        {
            cm_validate_req := {
                signalType := p_signalType,
                vrVarField := {
                    pilot_timer := p_pilot_timer,
                    result := p_result
                }
            }
        }
    }
    template MME_Payload md_CMN_CMN_CmValidateCnf_001(
        in template(present) Result_TYPE p_result) := {

        {
            cm_validate_cnf := {
                signalType := '00'H,
                vcVarField := {
                    toggle_num := '00'H,
                    result := p_result
                }
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmValidateCnf_002(
        in template(present) ToggleNum_TYPE p_toggle_num,
        in template(present) Result_TYPE p_result) := {

        {
            cm_validate_cnf := {
                signalType := '00'H,
                vcVarField := {
                    toggle_num := p_toggle_num,
                    result := p_result
                }
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmValidateCnf_003(
        in template(present) SignalType_TYPE p_signalType,
        in template(present) ToggleNum_TYPE p_toggle_num,
        in template(present) Result_TYPE p_result) := {

        {
            cm_validate_cnf := {
                signalType := p_signalType,
                vcVarField := {
                    toggle_num := p_toggle_num,
                    result := p_result
                }
            }
        }
    }
}
```

F.1.6 CMN templates for CmSlacMatch

```
module Templates_CMN_CmSlacMatch {

   import from DataStructure_SLAC  all;

   template MME_Payload md_CMN_CMN_CmSlacMatchReq_001 (
        template(present) SLAC_Header   v_slac_header,
        template(present) MACAddress_TYPE v_pevmac,
        template(present) MACAddress_TYPE v_evsemac,
        template(present) RunID_TYPE  v_runid) := {

       payload := {
           cm_slac_match_req := {
                 slac_header := v_slac_header,
                 mvflength := '003E'H,
                 pevid := '0000000000000000000000000000000000'H,
                 pevmac := v_pevmac,
                 evseid := '0000000000000000000000000000000000'H,
                 evsemac := v_evsemac,
                 runid := v_runid,
                 res0 := '0000000000000000'H
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmSlacMatchReq_002 (
         template(present) SLAC_Header v_slac_header,
         template(present) Mvflength_TYPE v_mvflength,
         template(present) StationID_TYPE v_pevid,
         template(present) MACAddress_TYPE v_pevmac,
         template(present) StationID_TYPE v_evseid,
         template(present) MACAddress_TYPE v_evsemac,
         template(present) RunID_TYPE  v_runid) := {

        payload := {
            cm_slac_match_req := {
                  slac_header := v_slac_header,
                  mvflength := v_mvflength,
                  pevid := v_pevid,
                  pevmac := v_pevmac,
                  evseid := v_evseid,
                  evsemac := v_evsemac,
                  runid := v_runid,
                  res0 := '0000000000000000'H
             }
         }
    }


    template MME_Payload md_CMN_CMN_CmSlacMatchCnf_001 (

        template(present) SLAC_Header   v_slac_header,
        template(present) MACAddress_TYPE v_pevmac,
        template(present) MACAddress_TYPE v_evsemac,
        template(present) RunID_TYPE  v_runid,
        template(present) NID_TYPE v_nid,
        template(present) NMK_TYPE v_nmk) := {

       payload := {
           cm_slac_match_cnf := {
                   slac_header := v_slac_header,
                   mvflength := '0056'H,
                   pevid := '0000000000000000000000000000000000'H,
                   pevmac := v_pevmac,
                   evseid := '0000000000000000000000000000000000'H,
                   evsemac := v_evsemac,
                   runid := v_runid,
                   res0 := '0000000000000000'H,
                   nid := v_nid,
                   res1 := '00'H,
                   nmk := v_nmk
                 }
           }
      }

    template MME_Payload md_CMN_CMN_CmSlacMatchCnf_002 (

        template(present) SLAC_Header v_slac_header,
        template(present) Mvflength_TYPE v_mvflength,
        template(present) StationID_TYPE v_pevid,
        template(present) MACAddress_TYPE v_pevmac,
        template(present) StationID_TYPE v_evseid,
        template(present) MACAddress_TYPE v_evsemac,
        template(present) RunID_TYPE  v_runid,
        template(present) NID_TYPE v_nid,
        template(present) NMK_TYPE v_nmk) := {

       payload := {
           cm_slac_match_cnf := {
                   slac_header := v_slac_header,
                   mvflength := v_mvflength,
                   pevid := v_pevid,
                   pevmac := v_pevmac,
                   evseid := v_evseid,
                   evsemac := v_evsemac,
                   runid := v_runid,
                   res0 := '0000000000000000'H,
                   nid := v_nid,
                   res1 := '00'H,
                   nmk := v_nmk
                 }
           }
      }
}
```

F.1.7 CMN templates for CmSetKey

```
module Templates_CMN_CmSetKey {

   import from DataStructure_SLAC  all;

    template MME_Payload md_CMN_CMN_CmSetKeyReq_001 (
        template(present) NID_TYPE v_nid,
        template(present) NewKey_TYPE v_neykey) := {

       payload := {
           cm_set_key_req := {
                   keytype := '01'H,
                    mynonce := '00000000'H,
                 yournonce := '00000000'H,
                 pid := '04'H,
                 prn := '0000'H,
                 pmn := '00'H,
                 ccocapability := '00'H,
                 nid := v_nid,
                 neweks := '01'H,
                 neykey := v_neykey
            }
        }
    }

    template MME_Payload mdw_CMN_CMN_CmSetKeyCnf_001(
        in template(present) Result_TYPE p_result) := {

       payload := {
           cm_set_key_cnf := {
               result := p_result,
               mynonce := ?,
               yournonce := ?,
               pid := ?,
               prn := ?,
               pmn := ?,
               ccocapability := ?
              }
           }
      }
}
```

F.1.8 CMN templates for CmAmpMap

```
module Templates_CMN_CmAmpMap {

    import from DataStructure_SLAC  all;

    template MME_Payload m_CMN_CMN_CmAmpMapReq_001() := {
        {
            cm_amp_map_req := {
                amlen := '0395'H,
                listAmdata := {
                    amdata := {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
```

```
                    'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                    'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                    '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                    'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                    'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                    'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                    '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                    'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                    'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                    'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                    '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                    'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                    'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                    'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                    '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                    'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                    '00'H} & {  '00'H} & {  '00'H} & {  '00'H}
            }
        }
    }
}

template MME_Payload md_CMN_CMN_CmAmpMapReq_002(
    in template(present) Amlen_TYPE p_amlen,
    in template(present) ListofAmdata_TYPE p_listAmdata) := {

    {
        cm_amp_map_req := {
                amlen := p_amlen,
                listAmdata := p_listAmdata
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmAmpMapReq_003(
        in template (present) Amlen_TYPE p_amlen) := {
        {
            cm_amp_map_req := {
                amlen := p_amlen,
                listAmdata := {
                    amdata := {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {  'C0'H} & {
                        'BC'H} & {  'BC'H} & {  'AA'H} & {  'AA'H} & {  '8F'H} & {
                        '8F'H} & {  '8F'H} & {  '8F'H} & {  'A1'H} & {  'A1'H} & {
                        'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {  'A1'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
```

```
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {  '00'H} & {
                        '00'H} & {  '00'H} & {  '00'H} & {  '00'H}
                }
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmAmpMapCnf_001(
        in template (present) Result_TYPE p_result) := {

        {
            cm_amp_map_cnf := {
                result := p_result
            }
        }
    }
}
```

F.1.9 CMN templates for CmNwStats

```
module Templates_CMN_CmNwStats {

    import from DataStructure_SLAC  all;

    template MME_Payload md_CMN_CMN_CmNwStatsReq_001() := {
        {
            cm_nw_stats_req := {}
            }
    }

    template MME_Payload md_CMN_CMN_CmNwStatsCnf_001() := {
        {
            cm_nw_stats_cnf := {
                numStas := ?,
                listOfStas := ?
            }
        }
    }

    template MME_Payload md_CMN_CMN_CmNwStatsCnf_002() := {
        {
            cm_nw_stats_cnf := {
                numStas := '00'H,
                listOfStas := omit
            }
        }
    }
}
```

F.2 SECC + PLC bridge templates

This subclause includes all templates *specifications* where the EVSE is defined as SUT.

F.2.1 SECC templates for CmAttenCharInd

```
module Templates_SECC_CmAttenCharInd {

    import from DataStructure_SLAC  all;

    template MME_Payload mdw_SECC_CMN_CmAttenCharInd_001(
         template(present) SLAC_Header v_slac_header,
         template(present) MACAddress_TYPE v_source_address,
         template(present) RunID_TYPE v_runid,
         template(present) NumSounds_TYPE v_num_sounds):= {

         payload:= {
             cm_atten_char_ind := {
             slac_header := v_slac_header,
             source_address := v_source_address,
             runid := v_runid,
             source_id := '0000000000000000000000000000000000'H,
             resp_id := '0000000000000000000000000000000000'H,
             num_sounds := v_num_sounds,
             num_groups := '3A'H,
             attenuation_list := {
                 attenuation := {
                     ?, ?, ?, ?, ?, ?, ?, ? ,?, ?, ?, ?, ?, ?, ?, ?,
                     ?, ?, ?, ?, ?, ?, ?, ? ,?, ?, ?, ?, ?, ?, ?, ?,
                     ?, ?, ?, ?, ?, ?, ?, ? ,?, ?, ?, ?, ?, ?, ?, ?,
                     ?, ?, ?, ?, ?, ?, ?, ? ,?, ?}
                 }
             }
         }
     }

}
```

F.3 EVCC + PLC bridge templates

This subclause includes all templates *specifications* where the EV is defined as SUT.

F.3.1 EVCC templates for CmAttenProfileInd

```
module Templates_EVCC_CmAttenProfileInd {

  import from DataStructure_SLAC  all;

  template MME_Payload md_EVCC_CMN_CmAttenProfileInd_001(
        template(present) MACAddress_TYPE v_pev_address,
        template(present) NumGroups_TYPE v_num_groups,
        template AttenProfile_TYPE   v_attenuation_list):= {

        payload:= {
            cm_atten_profile_ind := {
                pev_address := v_pev_address,
               num_groups := v_num_groups,
               res1 := '00'H,
               attenuation_list := v_attenuation_list}
         }
  }

  template AttenProfile_TYPE mw_EVCC_CMN_AttenProfile_001() := {
        attenuation :={
        ?, ?, ?, ?, ?, ?, ?, ? ,?, ?, ?, ?, ?, ?, ?, ?,
        ?, ?, ?, ?, ?, ?, ?, ? ,?, ?, ?, ?, ?, ?, ?, ?,
        ?, ?, ?, ?, ?, ?, ?, ? ,?, ?, ?, ?, ?, ?, ?, ?,
        ?, ?, ?, ?, ?, ?, ?, ? ,?, ?}
  }
}
```

F.3.2 EVCC templates for CmAttenCharInd

```
module Templates_EVCC_CmAttenCharInd {

    import from DataStructure_SLAC  all;

    template MME_Payload md_EVCC_CMN_CmAttenCharInd_001(
         template(present) SLAC_Header v_slac_header,
         template(present) MACAddress_TYPE v_source_address,
         template(present) RunID_TYPE v_runid,
         template(present) NumSounds_TYPE v_num_sounds,
         template(present) AttenProfile_TYPE v_atten_list):= {

         payload:= {
             cm_atten_char_ind := {
             slac_header := v_slac_header,
             source_address := v_source_address,
             runid := v_runid,
             source_id := '0000000000000000000000000000000000'H,
             resp_id := '0000000000000000000000000000000000'H,
             num_sounds := v_num_sounds,
             num_groups :='3A'H,
             attenuation_list := v_atten_list
             }
         }
     }

    template MME_Payload md_EVCC_CMN_CmAttenCharInd_002(
         template(present) SLAC_Header v_slac_header,
         template(present) MACAddress_TYPE v_source_address,
         template(present) RunID_TYPE v_runid,
         template(present) NumSounds_TYPE v_num_sounds,
         template(present) StationID_TYPE v_source_id,
         template(present) StationID_TYPE v_resp_id,
         template(present) NumGroups_TYPE v_num_groups,
         template(present) AttenProfile_TYPE v_atten_list) := {

         payload:= {
             cm_atten_char_ind := {
             slac_header := v_slac_header,
             source_address := v_source_address,
             runid := v_runid,
             source_id := v_source_id,
             resp_id := v_resp_id,
             num_sounds := v_num_sounds,
             num_groups := v_num_groups,
             attenuation_list := v_atten_list
             }
        }
     }

    template AttenProfile_TYPE m_EVCC_CMN_atten_list_001() := {
        attenuation := {
               '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H,
               '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H,
               '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H,
               '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H,
               '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H,
               '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H,
               '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H, '1E'H,
               '1E'H, '1E'H
        }
    }

    template AttenProfile_TYPE m_EVCC_CMN_atten_list_002() := {
        attenuation := {
               '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H,
               '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H,
               '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H,
               '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H,
               '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H,
               '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H,
               '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H, '28'H,
               '28'H, '28'H
        }
    }
}
```

Annex G
(normative)

Data type definitions

G.1 Data types for PICS

```
module DataStructure_PICS_15118 {

    type enumerated ChargingMode {
        aC,
        dC
    }

    type enumerated IdentificationMode {
        pnC,
        eIM
    }

    type enumerated PlugType {
        type1,
        type2
    }

    type enumerated CableCapabilityACType {
        capability13A,
        capability20A,
        capability32A,
        capability63A,
        capability70A
    }

    type enumerated EIMDone {
        beforePlugin,
        afterPlugin,
        duringSlac,
        v2gAuthorization
    }

    type enumerated ZeroPow {
        sleepWithoutCharge,
        sleepAfterCharge,
        none_
    }
}
```

G.2 Data types for PIXIT

```
module DataStructure_PIXIT_15118_3 {

    type enumerated CmValidateEVCC {
         none_,
         cmValidate,
         unknown
    }

    type enumerated CmValidateSECC {
         none_,
         cmValidate
    }

    type enumerated DutyCycle {
         dc5,
         dc100
    }

    type enumerated ValidationFallbackHandling {
        continue_,
        skip,
        terminate,
        unknown
    }

    type enumerated ConcurrentValidationHandling {
        retry,
        iterate,
        unknown
    }

    type enumerated CLHandling {
        optionA,
        optionB
    }
}



module DataStructure_PIXIT_15118 {

    type enumerated Pause {
        pause,
        unknown,
        none_
    }
}
```

G.3 Data types for SLAC

```
module DataStructure_SLAC {

    type record MME{
        MME_Header mme_header,
        MME_Payload mme_payload
    }

    type record MME_Header{
        MMV_TYPE mmv,
        MMTYPE mmtype,
        Fmi_TYPE fmi optional,
        Fmsn_TYPE fmsn optional
    }

    type record MME_Payload{
        union  {
        CM_SLAC_PARM_REQ cm_slac_parm_req,
        CM_SLAC_PARM_CNF cm_slac_parm_cnf,
        CM_START_ATTEN_CHAR_IND cm_start_atten_char_ind,
        CM_ATTEN_CHAR_IND cm_atten_char_ind,
        CM_ATTEN_CHAR_RSP cm_atten_char_rsp,
        CM_MNBC_SOUND_IND cm_mnbc_sound_ind,
        CM_ATTEN_PROFILE_IND cm_atten_profile_ind,
        CM_VALIDATE_REQ cm_validate_req,
        CM_VALIDATE_CNF cm_validate_cnf,
        CM_SLAC_MATCH_REQ cm_slac_match_req,
        CM_SLAC_MATCH_CNF cm_slac_match_cnf,
        CM_SET_KEY_REQ cm_set_key_req,
        CM_SET_KEY_CNF cm_set_key_cnf,
        VS_PL_LNK_STATUS_REQ vs_pl_lnk_status_req,
        VS_PL_LNK_STATUS_CNF vs_pl_lnk_status_cnf,
        CM_AMP_MAP_REQ cm_amp_map_req,
        CM_AMP_MAP_CNF cm_amp_map_cnf,
        VS_HST_ACTION_REQ vs_hst_action_req,
        VS_HST_ACTION_RSP vs_hst_action_rsp,
        CM_NW_STATS_REQ cm_nw_stats_req,
        CM_NW_STATS_CNF cm_nw_stats_cnf
        } payload
     }

    type record SLAC_Header{
        hexstring application_type length(2),
        hexstring security_type length(2)
    }

    type hexstring MMV_TYPE length(2);
    type hexstring Fmi_TYPE length(2);
    type hexstring Fmsn_TYPE length(2);
    type hexstring OUI_TYPE length(6) with { variant "byteOrder=big-endian"};
type hexstring NMK_TYPE length(32) with { variant "byteOrder=big-endian"};
type hexstring NID_TYPE length(14) with { variant "byteOrder=big-endian"};
type hexstring MACAddress_TYPE length(12) with { variant "byteOrder=big-endian"};
type hexstring StationID_TYPE length(34);
type hexstring RunID_TYPE length(16);
type hexstring TimeOut_TYPE length(2);
type hexstring NumSounds_TYPE length(2);
type hexstring NumGroups_TYPE length(2);
type hexstring ToggleNum_TYPE length(2);
type hexstring RespType_TYPE length(2);
type hexstring Aag_TYPE length(2);
type hexstring Result_TYPE length(2);
type hexstring Mvflength_TYPE length(4);
type hexstring Res0_TYPE length(16);
type hexstring Res1_TYPE length(2);
type hexstring Count_TYPE length(2);
type hexstring SourceRnd_Type length(32);
type hexstring Attenuation_TYPE length(2);
type hexstring SignalType_TYPE length(2);
type hexstring PilotTimer_TYPE length(2);
type hexstring KeyType_TYPE length(2);
type hexstring MyNonce_TYPE length(8);
type hexstring YourNonce_TYPE length(8);
type hexstring PID_TYPE length(2);
type hexstring PRN_TYPE length(4);
type hexstring PMN_TYPE length(2);
type hexstring CCoCapability_TYPE length(2);
type hexstring NewEKS_TYPE length(2);
type hexstring NewKey_TYPE length(32) with { variant "byteOrder=big-endian"};
type hexstring LnkStatus_TYPE length(2);
type hexstring NumStas_TYPE length(2);
type hexstring DataRate_TYPE length(2);
type hexstring Amlen_TYPE length(4);
type hexstring Amdata_TYPE length(2);
type hexstring HostActionReq_TYPE length(2);
type hexstring SessionId_TYPE length(2);
type hexstring OutstandingRetries_TYPE length(4);
type hexstring RetryTimer10ms_TYPE length(4);
type hexstring MStatus_TYPE length(2);

type union MMTYPE{
    hexstring CM_SLAC_PARM_REQ ('6064'H),
    hexstring CM_SLAC_PARM_CNF ('6065'H),
    hexstring CM_START_ATTEN_CHAR_IND ('606A'H),
    hexstring CM_ATTEN_CHAR_IND ('606E'H),
    hexstring CM_ATTEN_CHAR_RSP ('606F'H),
    hexstring CM_MNBC_SOUND_IND ('6076'H),
    hexstring CM_VALIDATE_REQ ('6078'H),
    hexstring CM_VALIDATE_CNF ('6079'H),
    hexstring CM_SLAC_MATCH_REQ ('607C'H),
    hexstring CM_SLAC_MATCH_CNF ('607D'H),
    hexstring CM_ATTEN_PROFILE_IND ('6086'H),
    hexstring CM_SET_KEY_REQ ('6008'H),
    hexstring CM_SET_KEY_CNF ('6009'H),
    hexstring VS_PL_LNK_STATUS_REQ ('A0B8'H),
    hexstring VS_PL_LNK_STATUS_CNF ('A0B9'H),
    hexstring CM_AMP_MAP_REQ ('601C'H),
    hexstring CM_AMP_MAP_CNF ('601D'H),
    hexstring VS_HST_ACTION_REQ ('A062'H),
    hexstring VS_HST_ACTION_RSP    ('A063'H),
    hexstring CM_NW_STATS_REQ ('6048'H),
    hexstring CM_NW_STATS_CNF ('6049'H)
}

type record  CM_SLAC_PARM_REQ{
    SLAC_Header slac_header,
    RunID_TYPE runid
}

type record  CM_SLAC_PARM_CNF{
    MACAddress_TYPE msound_target,
    NumSounds_TYPE num_sounds,
    TimeOut_TYPE time_out,
    RespType_TYPE resp_type,
    MACAddress_TYPE forwarding_sta,
    SLAC_Header appheader,
    RunID_TYPE runid
}
```

```
type record CM_START_ATTEN_CHAR_IND{
    SLAC_Header slac_header,
    NumSounds_TYPE num_sounds,
    TimeOut_TYPE time_out,
    RespType_TYPE resp_type,
    MACAddress_TYPE forwarding_sta,
    RunID_TYPE runid
}

type record  CM_ATTEN_CHAR_IND{
    SLAC_Header slac_header,
    MACAddress_TYPE source_address,
    RunID_TYPE runid,
    StationID_TYPE source_id,
    StationID_TYPE resp_id,
    NumSounds_TYPE num_sounds,
    NumGroups_TYPE num_groups,
    AttenProfile_TYPE attenuation_list
}

type record AttenProfile_TYPE {
       record length (1 .. 58) of Attenuation_TYPE attenuation
}

type record Acvarfield_Type{
    MACAddress_TYPE source_address,
    RunID_TYPE runid,
    StationID_TYPE source_id,
    StationID_TYPE resp_id,
    Result_TYPE result
}

type record  CM_ATTEN_CHAR_RSP{
    SLAC_Header slac_header,
    Acvarfield_Type acvarfield
}

type record CM_MNBC_SOUND_IND{
    SLAC_Header slac_header,
    StationID_TYPE source_id,
    Count_TYPE count,
    RunID_TYPE runid,
    Res0_TYPE res0,
    SourceRnd_Type source_rnd
}

type record CM_ATTEN_PROFILE_IND{
    MACAddress_TYPE pev_address,
    NumGroups_TYPE num_groups,
    Res1_TYPE res1,
    AttenProfile_TYPE attenuation_list optional
}

type record  CM_VALIDATE_REQ{
    SignalType_TYPE signalType,
    VRVarField_TYPE vrVarField
}

type record CM_VALIDATE_CNF{
    SignalType_TYPE signalType,
    VCVarField_TYPE vcVarField
}

type record  CM_SLAC_MATCH_REQ{
    SLAC_Header slac_header,
    Mvflength_TYPE mvflength,
    StationID_TYPE pevid,
    MACAddress_TYPE pevmac,
    StationID_TYPE evseid,
    MACAddress_TYPE evsemac,
    RunID_TYPE runid,
    Res0_TYPE res0
}

type record  CM_SLAC_MATCH_CNF{
    SLAC_Header slac_header,
    Mvflength_TYPE mvflength,
    StationID_TYPE pevid,
        MACAddress_TYPE pevmac,
        StationID_TYPE evseid,
        MACAddress_TYPE evsemac,
        RunID_TYPE runid,
        Res0_TYPE res0,
        NID_TYPE nid,
        Res1_TYPE res1,
        NMK_TYPE nmk
    }

    type record CM_SET_KEY_REQ{
        KeyType_TYPE keytype,
        MyNonce_TYPE mynonce,
        YourNonce_TYPE yournonce,
        PID_TYPE pid,
        PRN_TYPE prn,
        PMN_TYPE pmn,
        CCoCapability_TYPE ccocapability,
        NID_TYPE nid,
        NewEKS_TYPE neweks,
        NewKey_TYPE neykey
    }

    type record CM_SET_KEY_CNF{
        Result_TYPE result,
        MyNonce_TYPE mynonce,
        YourNonce_TYPE yournonce,
        PID_TYPE pid,
        PRN_TYPE prn,
        PMN_TYPE pmn,
        CCoCapability_TYPE ccocapability
    }

    type record VS_PL_LNK_STATUS_REQ{
        OUI_TYPE oui
    } with {variant "FMI=false"}

    type record VS_PL_LNK_STATUS_CNF{
        OUI_TYPE oui,
        Result_TYPE result,
        LnkStatus_TYPE lnkStatus
    } with {variant "FMI=false"}

    type record VRVarField_TYPE {
        PilotTimer_TYPE pilot_timer,
        Result_TYPE result
    }

    type record VCVarField_TYPE {
        ToggleNum_TYPE toggle_num,
        Result_TYPE result
    }

    type record of MME ResponseMessageList_TYPE;

    type record MACAddressList_TYPE {
        record length (0 .. 10) of MACAddress_TYPE macAddressList optional
    }

    type float T_conn_comm_TYPE (0.0 .. 8.0);

    type record CM_AMP_MAP_REQ {
        Amlen_TYPE amlen,
        ListofAmdata_TYPE listAmdata
    }

    type record CM_AMP_MAP_CNF {
        Result_TYPE result
    }

    type record ListofAmdata_TYPE {
        record length (0 .. 1155) of Amdata_TYPE amdata
    }

    type record VS_HST_ACTION_REQ{
        OUI_TYPE oui,
        HostActionReq_TYPE hostActionReq,
        SessionId_TYPE sessionId,
        OutstandingRetries_TYPE outstandingRetries,
        RetryTimer10ms_TYPE retryTimer10ms

    } with {variant "FMI=false"}

    type record VS_HST_ACTION_RSP{
        OUI_TYPE oui,
        MStatus_TYPE mStatus,
        HostActionReq_TYPE hostActionReq,
        SessionId_TYPE sessionId,
        OutstandingRetries_TYPE outstandingRetries
    } with {variant "FMI=false"}

    type record CM_NW_STATS_REQ{}

    type record CM_NW_STATS_CNF{
        NumStas_TYPE numStas,
        ListofStas_TYPE listOfStas optional
    }

    type record ListofStas_TYPE {
        record length (1 .. 8) of Stas_TYPE stas
    }

    type record Stas_TYPE {
        MACAddress_TYPE macAddress,
        DataRate_TYPE avgPHYDR_tx,
        DataRate_TYPE avgPHYDR_rx
    }
}

with {
  encode "SLAC";
}
```

Bibliography

ITU-T X.290, *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications — General concepts (April 1995)*

ITU-T X.292, *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications — The Tree and Tabular Combined Notation (TTCN) (May 2002)*

ETSI ES 201 873-1 V4.6.1, *TTCN-3: Core Language (June 2014)*

ICS 43.120

Price based on 403 pages