MICROSAR Smart Charge Communication

Technical Reference

Smart Charge Communication (ISO15118 & DIN70121)

Version 9.06.00

Authors	Phanuel Hieber
Status	Released

Document Information

History

Author	Date	Version	Remarks
Fabian Eisele	2012-11-16	4.00.01	- ISO15118 DIS
Fabian Eisele	2014-01-17	5.00.00	- ISO15118 FDIS
Fabian Eisele	2014-01-31	5.01.00	- ISO15118 FDIS + DIN70121 RC6
Fabian Eisele	2014-07-28	6.00.00	- added sequence diagrams - added RAM usage of EXI streams/structs
Fabian Eisele	2015-02-18	6.02.00	- updated example sequences - added more detailed explanation about the handling of the S2
Fabian Eisele	2015-03-10	6.03.01	- updated API description
Fabian Eisele	2015-03-17	6.03.02	- updated API description
Fabian Eisele	2015-05-27	7.00.00	- updated API description - update to MSR4R12
Fabian Eisele	2016-02-05	8.00.00	- updated API description
Fabian Eisele	2016-12-08	9.01.00	- updated API description
Phanuel Hieber	2017-02-01	9.02.00	- removed TxConfirmation
Phanuel Hieber	2017-09-14	9.06.00	- Updated API description - Added CSM

Reference Documents

No.	Source	Title	Version
[1]	ISO	ISO/IEC 15118-2	IS
[2]	DIN	DIN 70121	2014-12
[3]	RFC2898	PKCS #5: Password-Based Cryptography Specification	2

Scope of the Document

This technical reference describes the general use of the **SCC** basis software.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.xx	Created
4.00.xx	Added support for the DIS of ISO15118
5.00.xx	Update from DIS to FDIS of ISO15118
5.01.xx	Added support for DIN70121 RC6
5.02.xx	Added support for additional OEM
5.03.xx	Added support for Inductive Charging according to customer schema
6.00.xx	Added support for encryption of Private Keys Buffer optimization
6.02.xx	Improved support of IEC 61851-1 related handling
6.03.xx	Added support for SLAC handling
7.00.xx	Update to ASR4.2.1 (change in TcpIp API)
8.00.xx	Introduction of Scc_ReturnType Source of configuration is now configurable (ASR4 only)
8.01.xx	Introduced EMAID validation
9.00.xx	Update to support MSR4R15 API of TcpIp

Table 1 Component history

2 Introduction

This document describes the functionality, API and configuration of the MICROSAR BSW module **SCC**.

Supported AUTOSAR Release:	4.3+	
Supported Configuration Variants:	pre-compile	
Vendor ID:	**SCC**_VENDOR_ID	30 decimal (= Vector Informatik, according to HIS)
Module ID:	**SCC**_MODULE_ID	**255** decimal (according to ref.)

2.1 Architecture Overview

The following figure shows where the **SCC** is located in the AUTOSAR architecture.

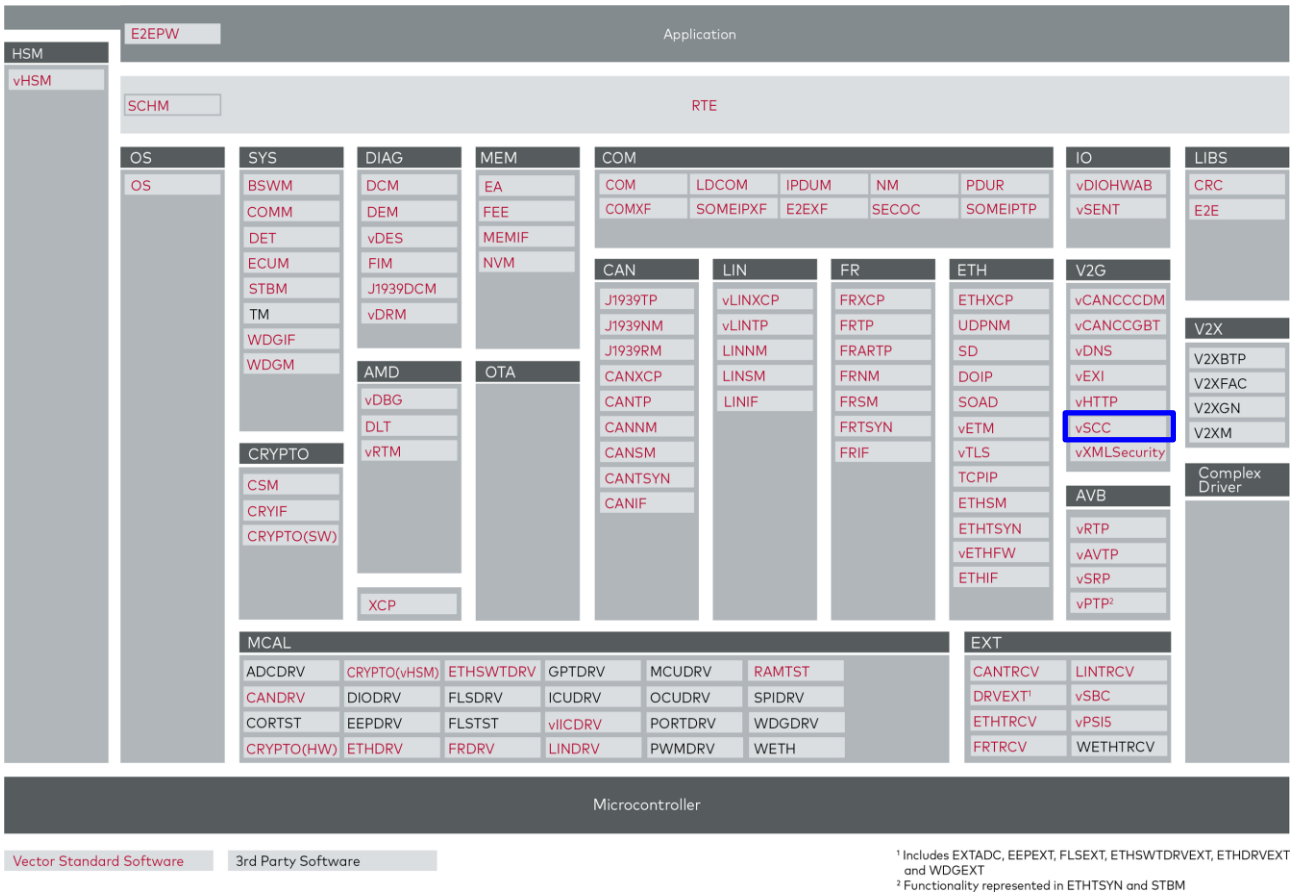


Figure 2-1 AUTOSAR Architecture Overview

The next figure shows the interfaces to adjacent modules of the **SCC**. These interfaces are described in chapter 5.

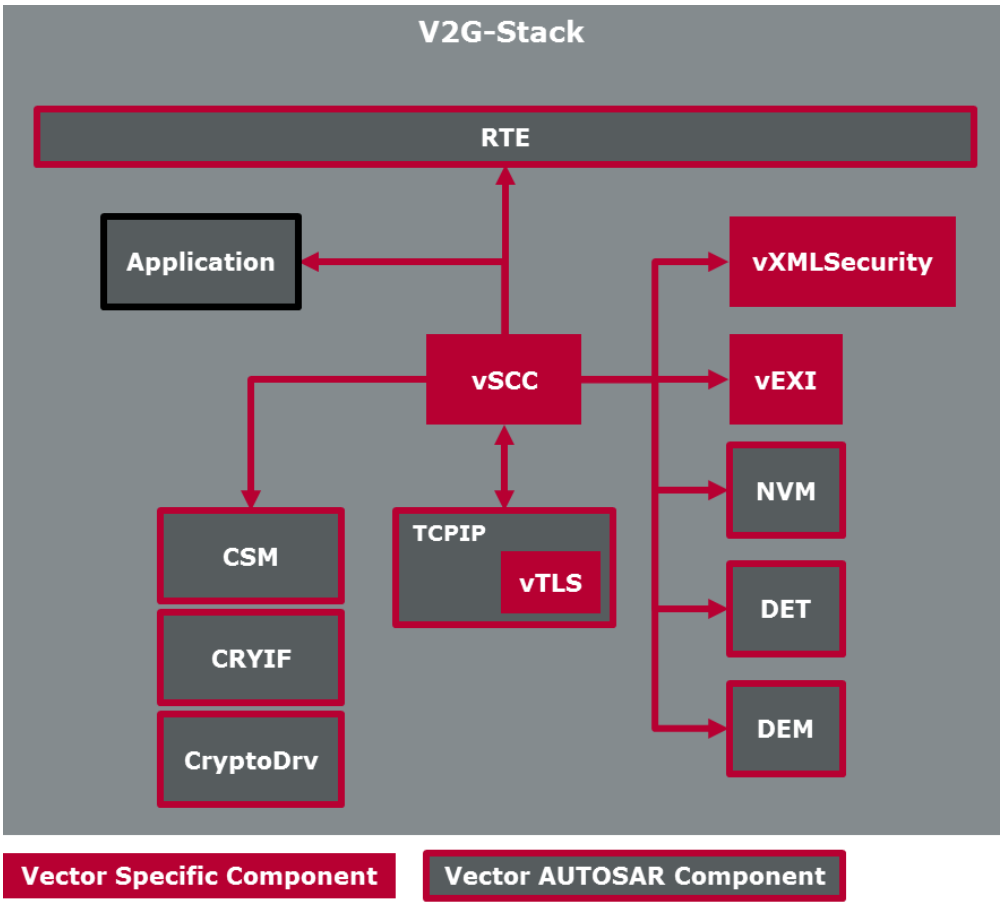


Figure 2-2 Interfaces to adjacent modules of the **SCC**

3 Functional Description

3.1 Features

The features listed in this chapter cover the complete functionality specified in [1].

Currently only the configuration variant 1, "pre-compile", is supported.

3.2 Initialization

The **SCC** component gets initialized by sequence of `Scc_InitMemory` followed by `Scc_Init` out of EcuM.

If EcuM is not used the application has to call `Scc_InitMemory` followed by `Scc_Init`.

The meaning of the parameter in `Scc_Init` depends on the selected configuration variant.

In case XmlSecurity is used, it needs to be initialized before **SCC**.

3.2.1 Configuration Variants 1 and 2 (Pre-Compile and Link-Time Configuration)

At variant 1 (Pre-compile Configuration) and Variant 2 (Link-Time Configuration) the pointer given to `Scc_Init` is ignored. At these configuration variants **SCC** is configured at compile or link time and has direct access to all configuration data which is stored in the files Scc_Cfg.h and Scc_Lcfg.c.

Example

```
Scc_Init(NULL_PTR);
```

Figure 3-1 Initialization in configuration variant 1 and variant 2

3.2.2 Configuration Variant 3 (Post-build Configuration)

In this configuration variant, the **SCC** component has to be initialized using the `Scc_Init` function with the address of the post-build configuration data passed as parameter. The declaration of the post-build configuration data is contained in the file Scc_PBcfg.h and Scc_PBcfg.c.

Example

```
Scc_Init(&Scc_Config);
```

Figure 3-2 Initialization in configuration variant 3

3.2.3 Multi Config (Pre-compile configuration)

SCC supports multiple configurations of the CAN channel. Therefore the pointer given to `Scc_Init` contains the reference to one of the multiple structs, which contains the CAN parameters for the selected configuration.

3.3 States

The **SCC** component is operational after initialization.

3.4 Main Functions

The **SCC** Main Function shall be called by the Schedule Manager SchM.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()`, if development error reporting is enabled (i.e. pre-compile parameter **SCC**`_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported **SCC** ID is **255**.

The reported service IDs identify the services which are described in chapter 5.2. To allow easier debugging, there are also service IDs for internal functions. The service IDs can be found in the Scc.h.

3.5.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()`, if production error reporting is not disabled (i.e. pre-compile parameter **SCC**`_VDEM_ERROR_DETECT==STD_OFF`).

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

Error Code	Description
SCC_DEM_EXI	An error occurred while encoding or decoding an EXI message.
SCC_DEM_UNEXPECTED_MSG	An unexpected message or message element was received by the EV.
SCC_DEM_XML_SEC	An error occurred while validating or generating a XmlSecurity signature.
SCC_DEM_CRYPTO	An error occurred while decrypting the encrypted private key.
SCC_DEM_IP_BASE	An error occurred while using a function provided by the component IpBase.
SCC_DEM_NVM_READ_ CONTR_CERT_CHAIN_SIZE_FAIL	The NvM block of the Contract Certificate chain size could not be read.
SCC_DEM_NVM_READ_CONTR_CERT_FAIL	An NvM block of one of the contract certificates could not be read.

SCC_DEM_NVM_READ_ CONTR_CERT_PRIV_KEY_FAIL	An NvM block of one of the private keys of the contract certificates could not be read.
SCC_DEM_NVM_READ_ CONTR_SUB_CERT_FAIL	An NvM block of one of the contract sub certificates could not be read.
SCC_DEM_NVM_READ_PROV_CERT_FAIL	An NvM block of one of the provisioning certificates could not be read.
SCC_DEM_NVM_READ_ PROV_CERT_PRIV_KEY_FAIL	An NvM block of one of the private keys of the provisioning certificates could not be read.
SCC_DEM_NVM_READ_ROOT_CERT_FAIL	An NvM block of one of the root certificates could not be read.
SCC_DEM_SAP_NO_NEGOTIATION	The EVSE does not support any of the EV's schemas.

Table 2 Errors reported to DEM

4 Integration

This chapter gives necessary information for the integration of the **SCC** into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the **SCC** contains following files:

4.1.1 Static Files

File Name	Description
Scc.c	Static source for core.
Scc.h	Static header for API.
Scc_Priv.h	Static header for internal macro and variable declaration.
Scc_Priv.c	Static source for internal helper functions.
Scc_StateM_Vector.h	Static header for internal state machine.
Scc_StateM_Vector.c	Static source for internal state machine.
Scc_Exi.h	Static header for EXI parser.
Scc_Exi.c	Static source for EXI parser.
Scc_ExiRx_ISO.c	Static source for EXI parser for ISO15118 messages.
Scc_ExiTx_ISO.c	Static source for EXI parser for ISO15118 messages.
Scc_ExiRx_DIN.c	Static source for EXI parser for DIN70121 messages.
Scc_ExiTx_DIN.c	Static source for EXI parser for DIN70121 messages.
Scc_Cbk.h	Static header for TcpIp call-back API.
Scc_Types.h	Static header for type definitions.

Table 3 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool [config tool].

File Name	Description
Scc_Cfg.h	Generated header file for pre-compile time configuration data.
Scc_Interface_Cfg.h	Generated header for parameter mappings.
Scc_ConfigParams_Cfg.h	Generated header for configuration parameter mappings.
Scc_Lcfg.h	Generated header for link time configuration data.
Scc_Lcfg.c	Generated source for link time configuration data.
Scc_PBcfg.h	Generated header for post-build time configuration data.
Scc_PBcfg.c	Generated source for post-build time configuration data.

Table 4 Generated files

4.1.3 Template Files

File Name	Description
_ApplScc_CbkStubs.c	Static source containing stub implementations of the callbacks. They can be used as templates for the implementation of the application callbacks.
_ApplScc_CbkStubs.h	Static header containing stub declarations. They can be used as templates for the implementation of the application callbacks.

Table 5 Template files

4.2 Include Structure

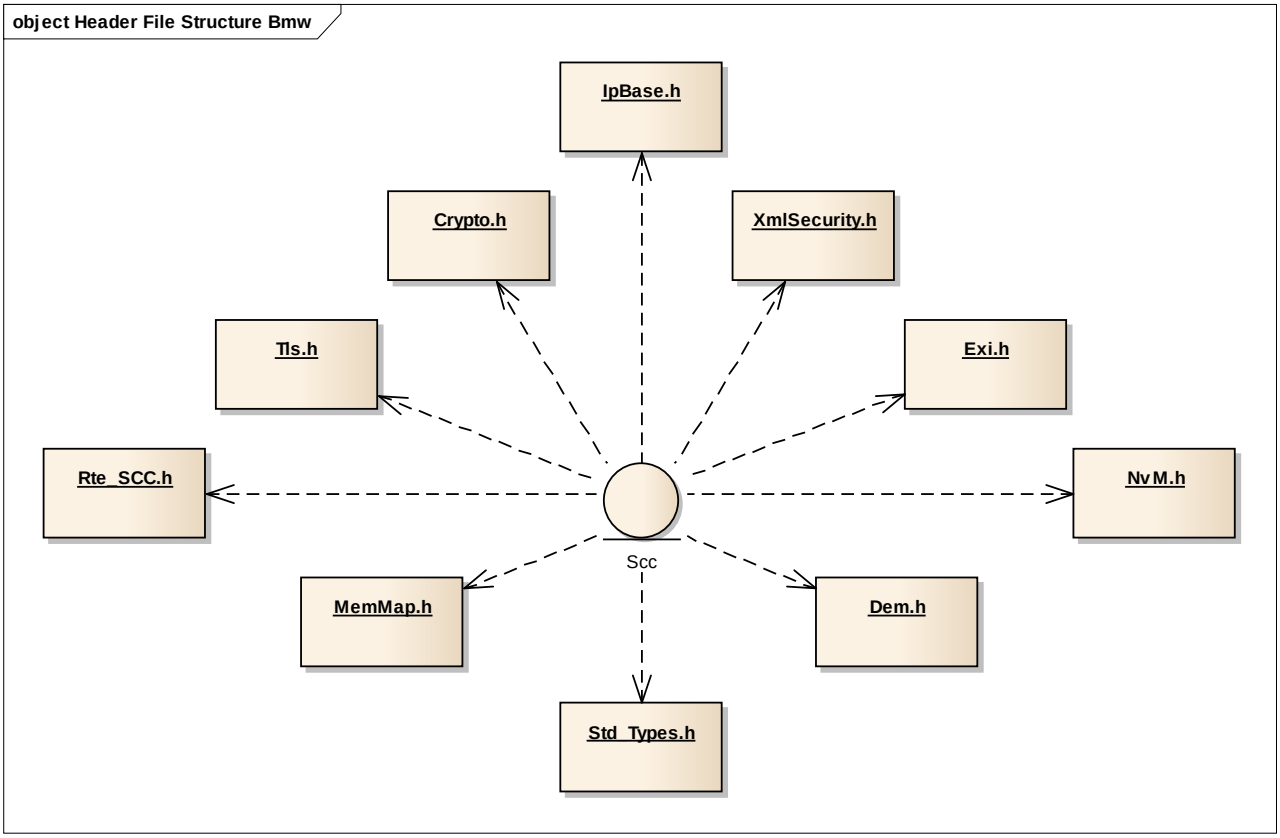


Figure 4-1 Include structure

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions of the **SCC** and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	SCC_VAR_NOINIT	SCC_VAR_ZERO_INIT	SCC_CONST	SCC_CODE	SCC_PBCFG
SCC_START_SEC_CODE SCC_STOP_SEC_CODE				■	
SCC_START_SEC_CONST_UNSPECIFIED SCC_STOP_SEC_CONST_UNSPECIFIED			■		
SCC_START_SEC_CONST_32BIT SCC_STOP_SEC_CONST_32BIT			■		
SCC_START_SEC_CONST_16BIT SCC_STOP_SEC_CONST_16BIT			■		
SCC_START_SEC_CONST_8BIT SCC_STOP_SEC_CONST_8BIT			■		
SCC_START_SEC_VAR_NOINIT_UNSPECIFIED SCC_STOP_SEC_VAR_NOINT_UNSPECIFIED	■				
SCC_START_SEC_VAR_NOINIT_32BIT SCC_STOP_SEC_VAR_NOINT_32BIT	■				
SCC_START_SEC_VAR_NOINIT_16BIT SCC_STOP_SEC_VAR_NOINT_16BIT	■				
SCC_START_SEC_VAR_NOINIT_8BIT SCC_STOP_SEC_VAR_NOINT_8BIT	■				
SCC_START_SEC_VAR_ZERO_INIT_UNSPECIFIED SCC_STOP_SEC_VAR_ ZERO_INIT _UNSPECIFIED		■			
SCC_START_SEC_VAR_ ZERO_INIT _32BIT SCC_STOP_SEC_VAR_ ZERO_INIT _32BIT		■			
SCC_START_SEC_VAR_ ZERO_INIT _16BIT SCC_STOP_SEC_VAR_ ZERO_INIT _16BIT		■			
SCC_START_SEC_VAR_ ZERO_INIT _8BIT SCC_STOP_SEC_VAR_ ZERO_INIT _8BIT		■			

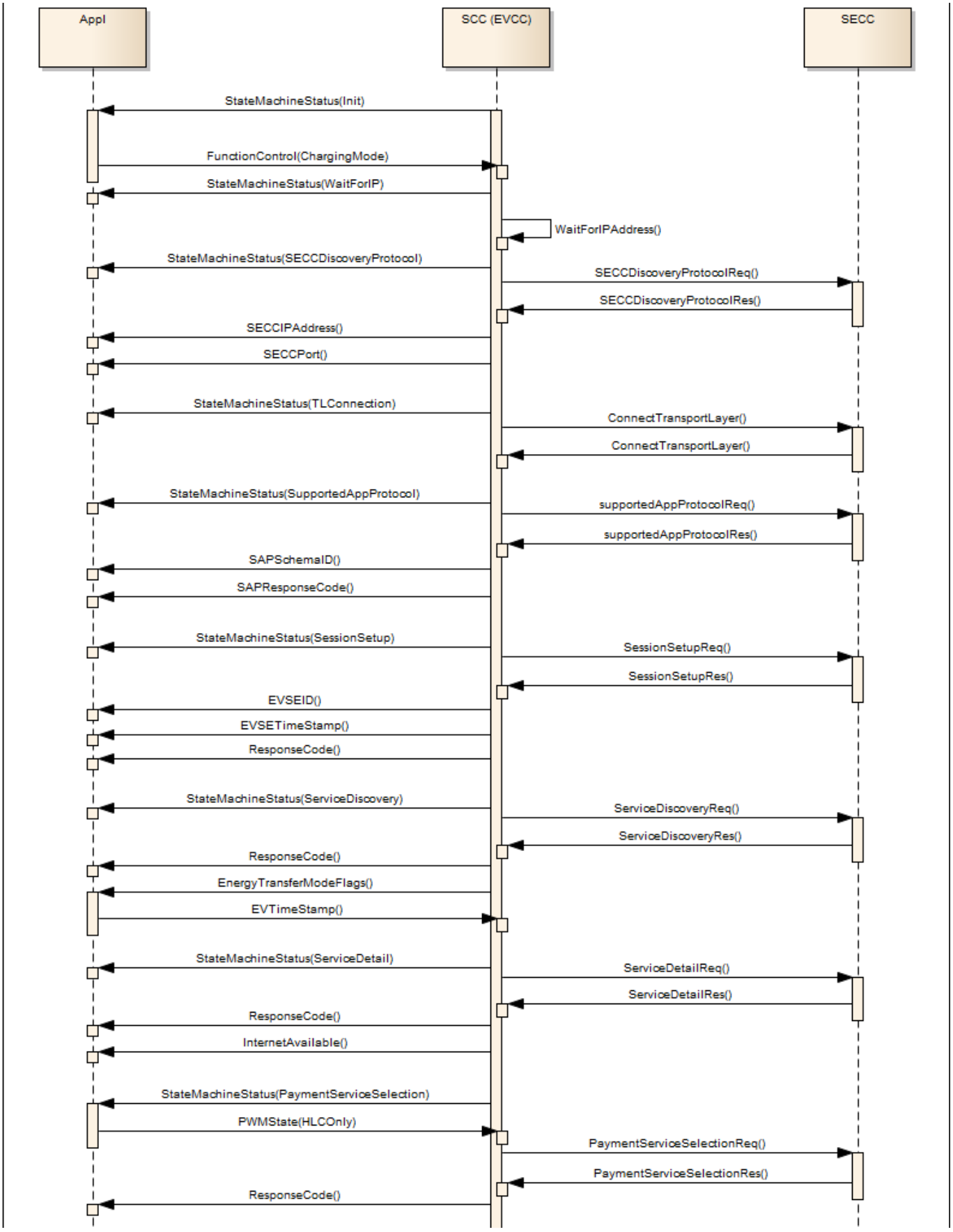
Table 6 Compiler abstraction and memory mapping

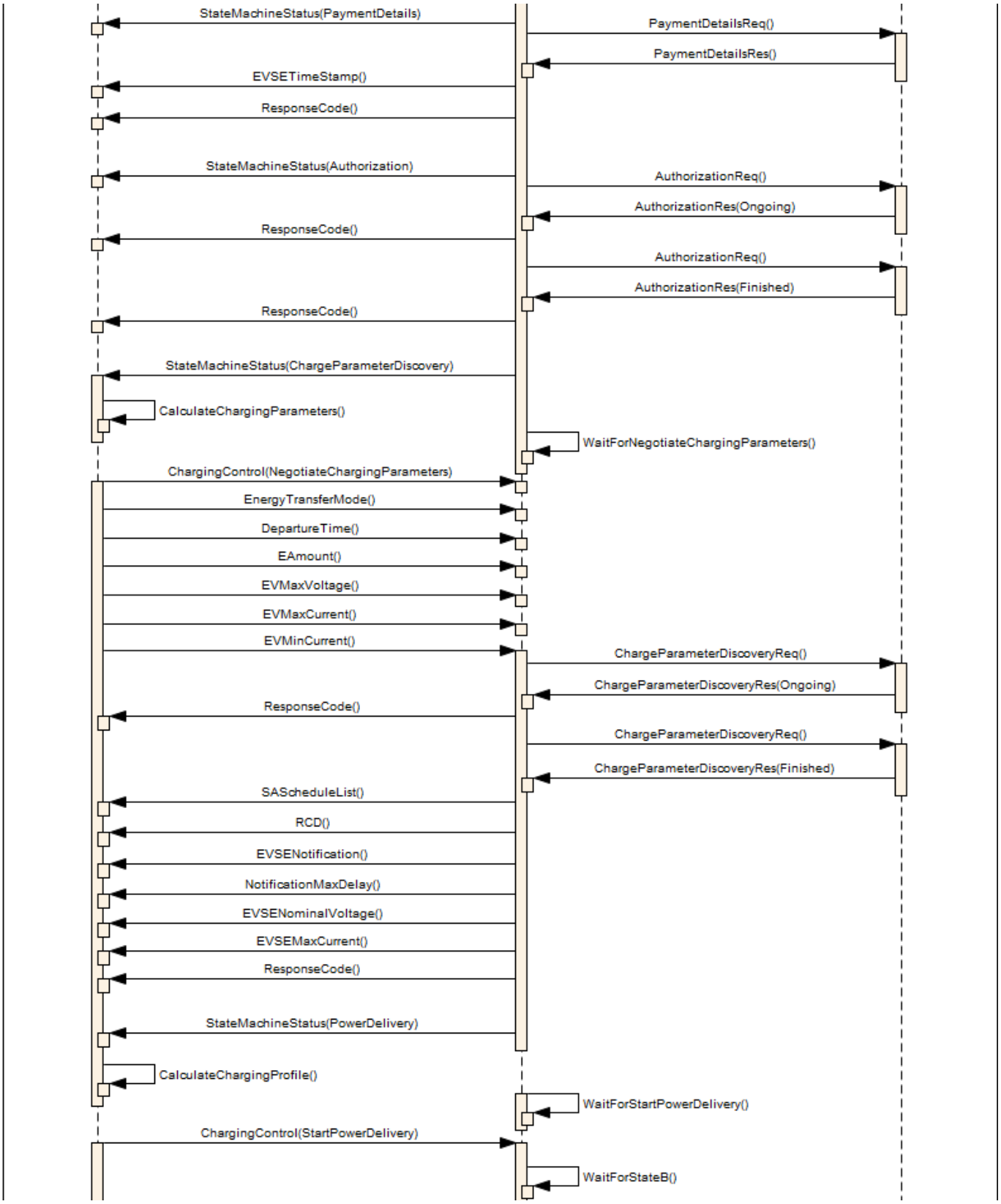
4.4 Example Sequences

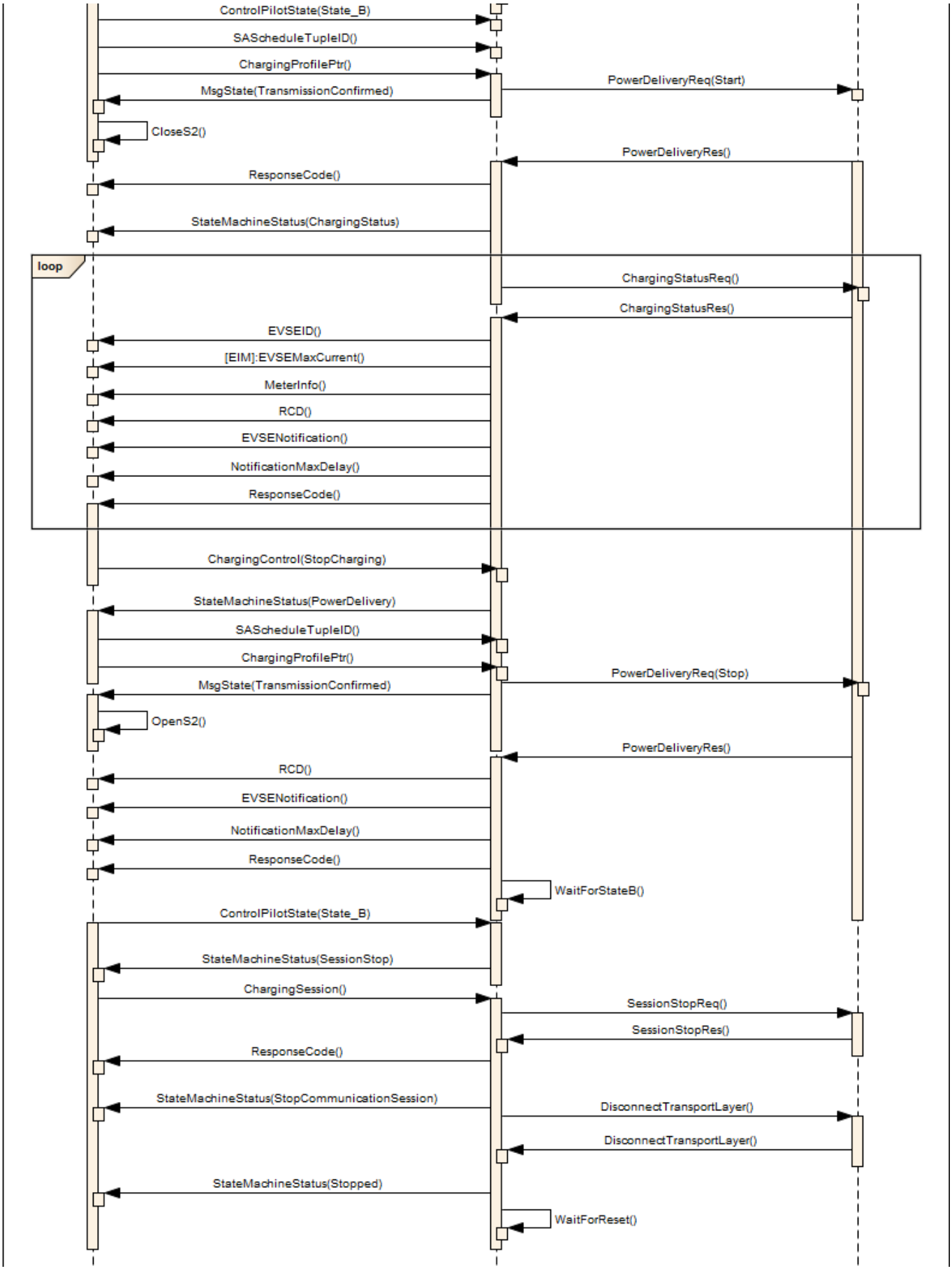
SCC can be used in a lot of different setups. One setup could be with according to ISO15118, with an enabled State Machine, EIM profile only and AC Charging. Another could be charging according to DIN70121. How **SCC** can be controlled in these use cases can be seen in the sequence diagrams below.

Note: The direction of the arrows only shows the data flow. The data will always be requested by **SCC** via a callback.

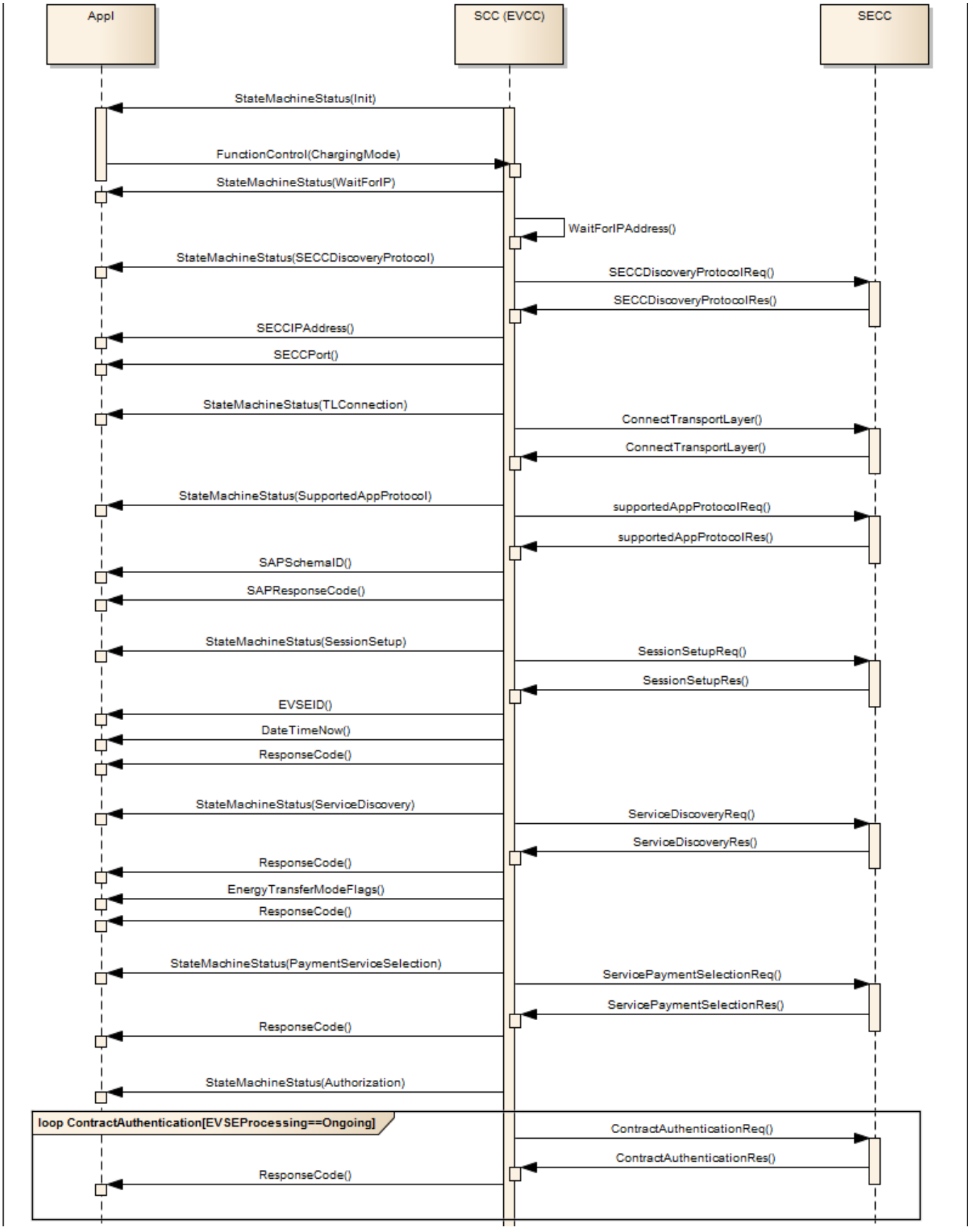
4.4.1 ISO AC

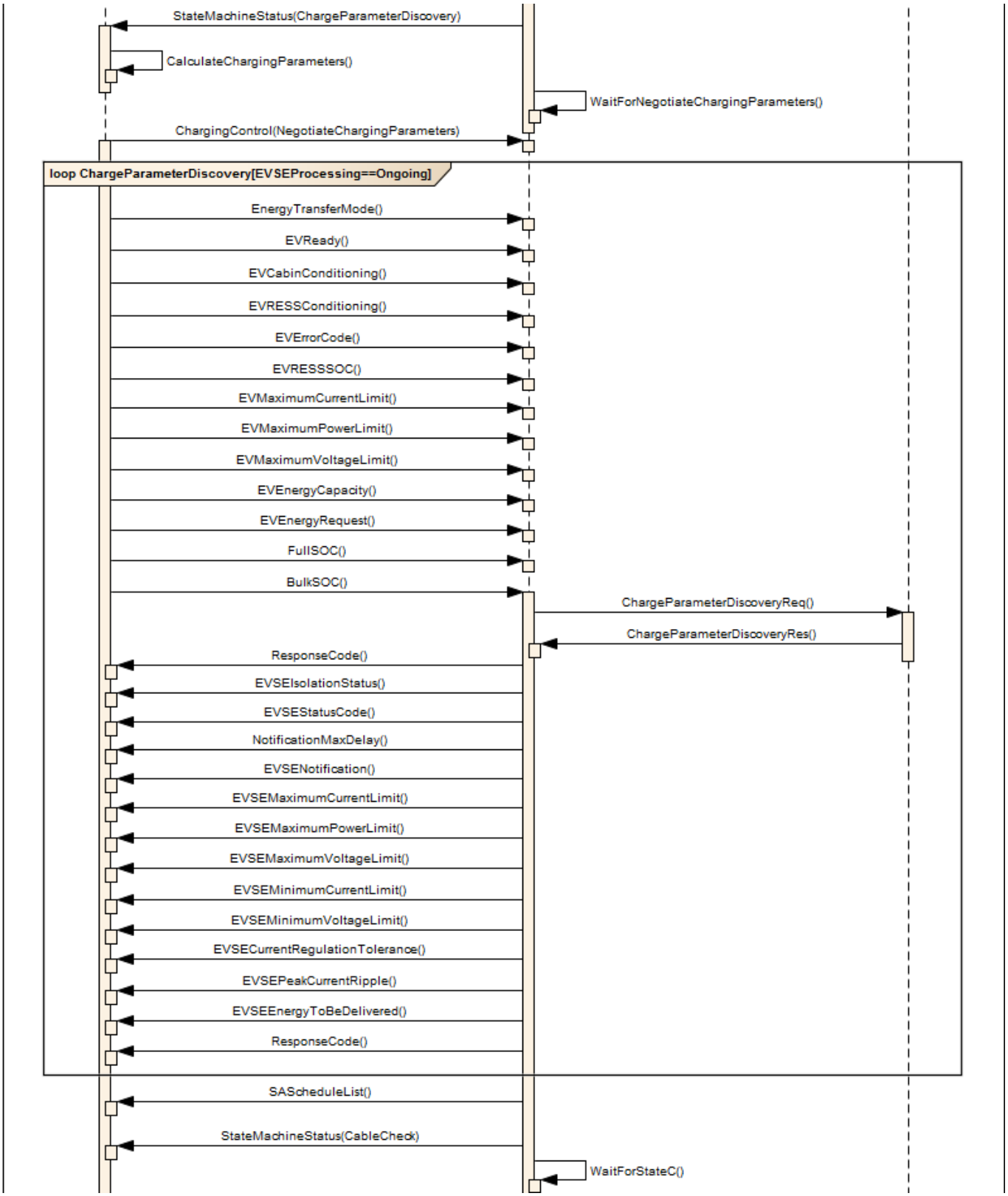

Appl
SCC (EVCC)
SECC
StateMachineStatus(Init)
FunctionControl(ChargingMode)
StateMachineStatus(WaitForIP)
WaitForIPAddress()
StateMachineStatus(SECCDiscoveryProtocol)
SECCDiscoveryProtocolReq()
SECCDiscoveryProtocolRes()
SECCIPAddress()
SECCPort()
StateMachineStatus(TLConnection)
ConnectTransportLayer()
ConnectTransportLayer()
StateMachineStatus(SupportedAppProtocol)
supportedAppProtocolReq()
supportedAppProtocolRes()
SAPSchemaID()
SAPResponseCode()
StateMachineStatus(SessionSetup)
SessionSetupReq()
SessionSetupRes()
EVSEID()
EVSETimeStamp()
ResponseCode()
StateMachineStatus(ServiceDiscovery)
ServiceDiscoveryReq()
ServiceDiscoveryRes()
ResponseCode()
EnergyTransferModeFlags()
EVTimeStamp()
StateMachineStatus(ServiceDetail)
ServiceDetailReq()
ServiceDetailRes()
ResponseCode()
InternetAvailable()
StateMachineStatus(PaymentServiceSelection)
PWMState(HLCOnly)
PaymentServiceSelectionReq()
PaymentServiceSelectionRes()
ResponseCode()

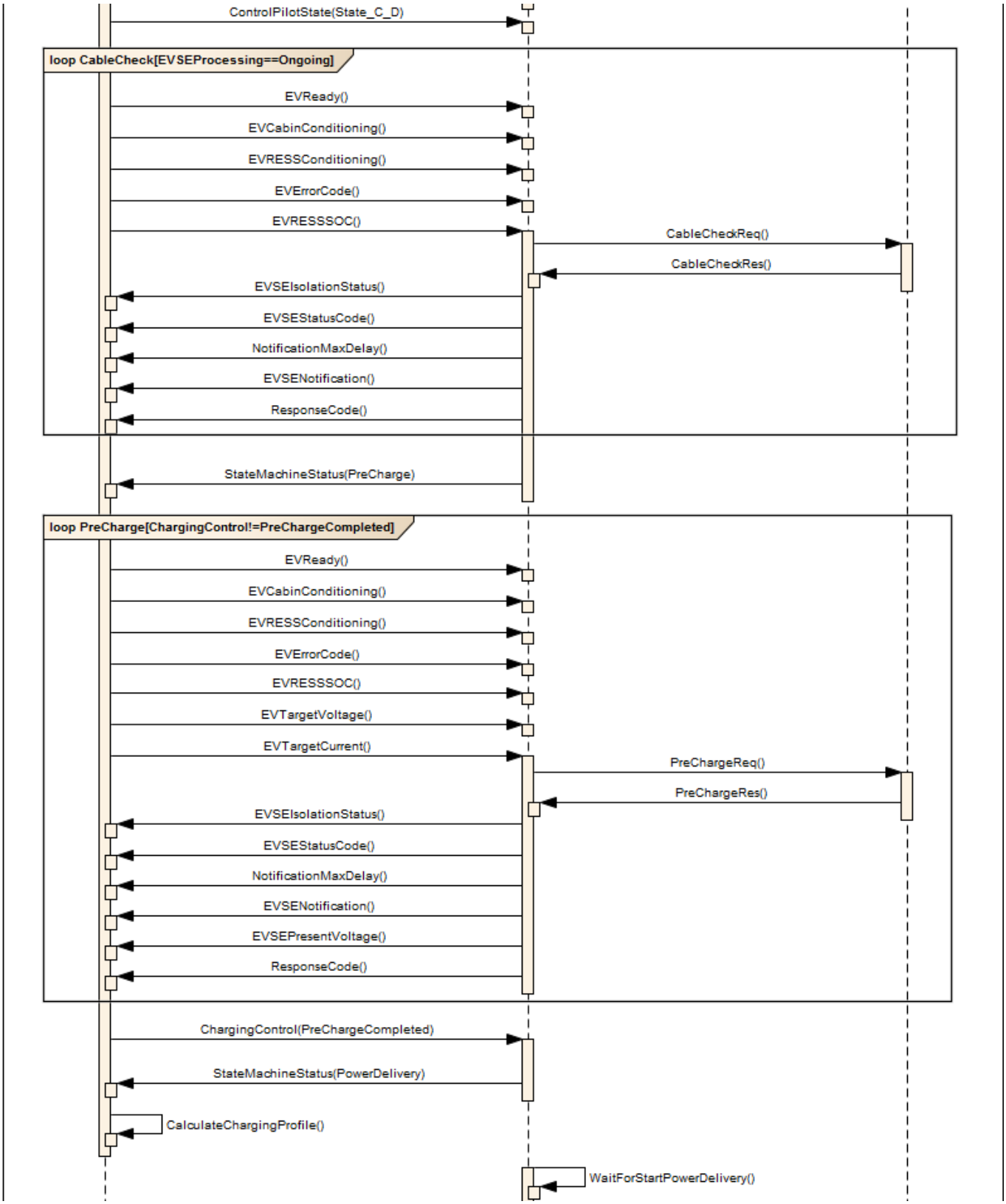

StateMachineStatus(PaymentDetails)
PaymentDetailsReq()
PaymentDetailsRes()
EVSETimeStamp()
ResponseCode()
StateMachineStatus(Authorization)
AuthorizationReq()
AuthorizationRes(Ongoing)
ResponseCode()
AuthorizationReq()
AuthorizationRes(Finished)
ResponseCode()
StateMachineStatus(ChargeParameterDiscovery)
CalculateChargingParameters()
WaitForNegotiateChargingParameters()
ChargingControl(NegotiateChargingParameters)
EnergyTransferMode()
DepartureTime()
EAmount()
EVMaxVoltage()
EVMaxCurrent()
EVMinCurrent()
ChargeParameterDiscoveryReq()
ChargeParameterDiscoveryRes(Ongoing)
ResponseCode()
ChargeParameterDiscoveryReq()
ChargeParameterDiscoveryRes(Finished)
SAScheduleList()
RCD()
EVSENotification()
NotificationMaxDelay()
EVSENominalVoltage()
EVSEMaxCurrent()
ResponseCode()
StateMachineStatus(PowerDelivery)
CalculateChargingProfile()
WaitForStartPowerDelivery()
ChargingControl(StartPowerDelivery)
WaitForStateB()

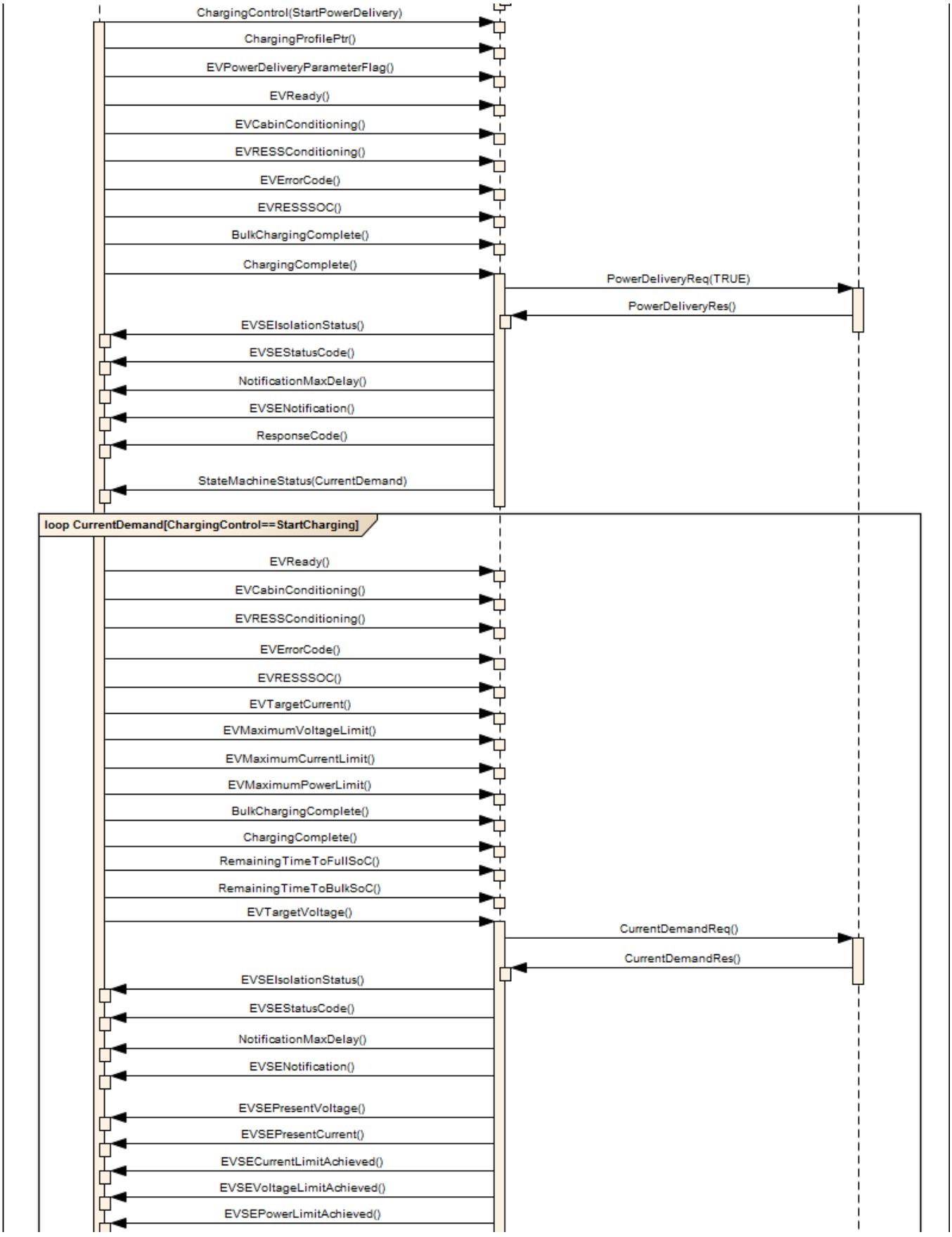

ControlPilotState(State_B)
SAScheduleTupleID()
ChargingProfilePtr()
MsgState(TransmissionConfirmed)
PowerDeliveryReq(Start)
CloseS2()
PowerDeliveryRes()
ResponseCode()
StateMachineStatus(ChargingStatus)
loop
ChargingStatusReq()
ChargingStatusRes()
EVSEID()
[EIM]:EVSEMaxCurrent()
MeterInfo()
RCD()
EVSENotification()
NotificationMaxDelay()
ResponseCode()
ChargingControl(StopCharging)
StateMachineStatus(PowerDelivery)
SAScheduleTupleID()
ChargingProfilePtr()
PowerDeliveryReq(Stop)
MsgState(TransmissionConfirmed)
OpenS2()
PowerDeliveryRes()
RCD()
EVSENotification()
NotificationMaxDelay()
ResponseCode()
WaitForStateB()
ControlPilotState(State_B)
StateMachineStatus(SessionStop)
ChargingSession()
SessionStopReq()
SessionStopRes()
ResponseCode()
StateMachineStatus(StopCommunicationSession)
DisconnectTransportLayer()
DisconnectTransportLayer()
StateMachineStatus(Stopped)
WaitForReset()

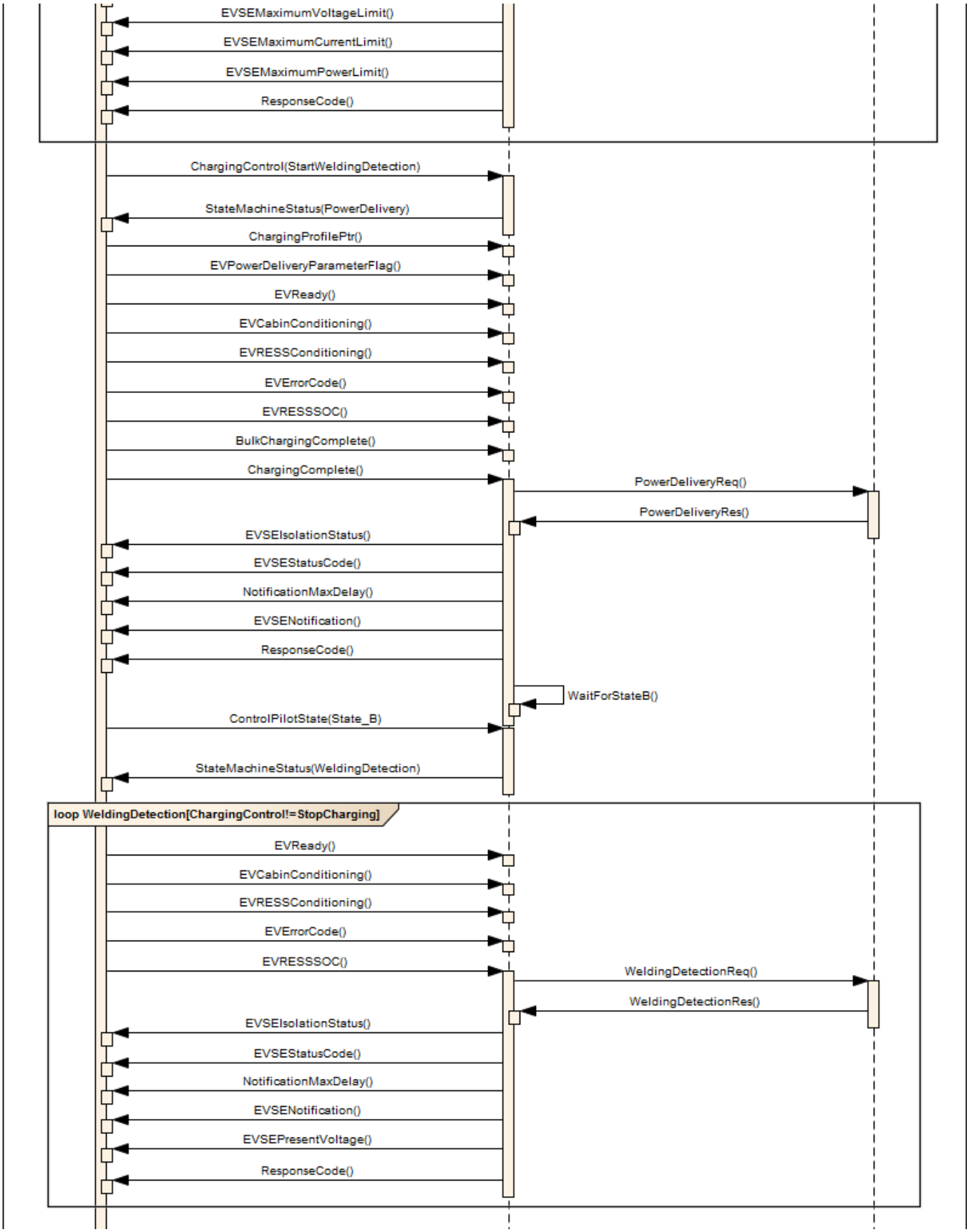
4.4.2 DIN

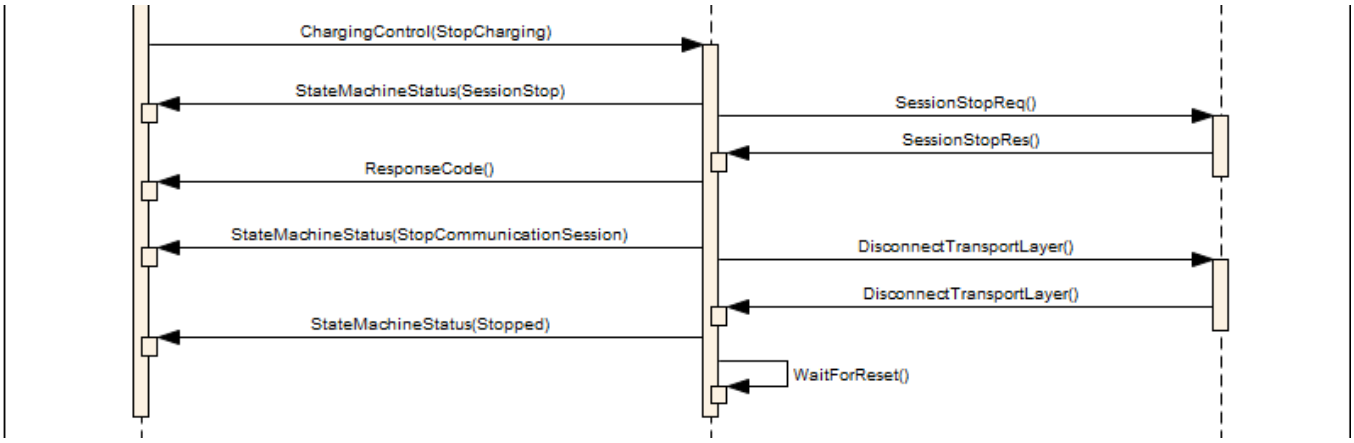

Appl
SCC (EVCC)
SECC
StateMachineStatus(Init)
FunctionControl(ChargingMode)
StateMachineStatus(WaitForIP)
WaitForIPAddress()
StateMachineStatus(SECCDiscoveryProtocol)
SECCDiscoveryProtocolReq()
SECCDiscoveryProtocolRes()
SECCIPAddress()
SECCPort()
StateMachineStatus(TLConnection)
ConnectTransportLayer()
ConnectTransportLayer()
StateMachineStatus(SupportedAppProtocol)
supportedAppProtocolReq()
supportedAppProtocolRes()
SAPSchemaID()
SAPResponseCode()
StateMachineStatus(SessionSetup)
SessionSetupReq()
SessionSetupRes()
EVSEID()
DateTimeNow()
ResponseCode()
StateMachineStatus(ServiceDiscovery)
ServiceDiscoveryReq()
ServiceDiscoveryRes()
ResponseCode()
EnergyTransferModeFlags()
ResponseCode()
StateMachineStatus(PaymentServiceSelection)
ServicePaymentSelectionReq()
ServicePaymentSelectionRes()
ResponseCode()
StateMachineStatus(Authorization)
loop ContractAuthentication[EVSEProcessing==Ongoing]
ContractAuthenticationReq()
ContractAuthenticationRes()
ResponseCode()


StateMachineStatus(ChargeParameterDiscovery)
CalculateChargingParameters()
WaitForNegotiateChargingParameters()
ChargingControl(NegotiateChargingParameters)
loop ChargeParameterDiscovery[EVSEProcessing==Ongoing]
EnergyTransferMode()
EVReady()
EVCabinConditioning()
EVRESSConditioning()
EVErrorCode()
EVRESSSOC()
EVMaximumCurrentLimit()
EVMaximumPowerLimit()
EVMaximumVoltageLimit()
EVEnergyCapacity()
EVEnergyRequest()
FullSOC()
BulkSOC()
ChargeParameterDiscoveryReq()
ChargeParameterDiscoveryRes()
ResponseCode()
EVSEIsolationStatus()
EVSEStatusCode()
NotificationMaxDelay()
EVSENotification()
EVSEMaximumCurrentLimit()
EVSEMaximumPowerLimit()
EVSEMaximumVoltageLimit()
EVSEMinimumCurrentLimit()
EVSEMinimumVoltageLimit()
EVSECurrentRegulationTolerance()
EVSEPeakCurrentRipple()
EVSEEnergyToBeDelivered()
ResponseCode()
SAScheduleList()
StateMachineStatus(CableCheck)
WaitForStateC()


ControlPilotState(State_C_D)
loop CableCheck[EVSEProcessing==Ongoing]
EVReady()
EVCabinConditioning()
EVRESSConditioning()
EVErrorCode()
EVRESSSOC()
CableCheckReq()
CableCheckRes()
EVSEIsolationStatus()
EVSEStatusCode()
NotificationMaxDelay()
EVSENotification()
ResponseCode()
StateMachineStatus(PreCharge)
loop PreCharge[ChargingControl!=PreChargeCompleted]
EVReady()
EVCabinConditioning()
EVRESSConditioning()
EVErrorCode()
EVRESSSOC()
EVTargetVoltage()
EVTargetCurrent()
PreChargeReq()
PreChargeRes()
EVSEIsolationStatus()
EVSEStatusCode()
NotificationMaxDelay()
EVSENotification()
EVSEPresentVoltage()
ResponseCode()
ChargingControl(PreChargeCompleted)
StateMachineStatus(PowerDelivery)
CalculateChargingProfile()
WaitForStartPowerDelivery()


ChargingControl(StartPowerDelivery)
ChargingProfilePtr()
EVPowerDeliveryParameterFlag()
EVReady()
EVCabinConditioning()
EVRESSConditioning()
EVErrorCode()
EVRESSSOC()
BulkChargingComplete()
ChargingComplete()
PowerDeliveryReq(TRUE)
PowerDeliveryRes()
EVSEIsolationStatus()
EVSEStatusCode()
NotificationMaxDelay()
EVSENotification()
ResponseCode()
StateMachineStatus(CurrentDemand)
loop CurrentDemand[ChargingControl==StartCharging]
EVReady()
EVCabinConditioning()
EVRESSConditioning()
EVErrorCode()
EVRESSSOC()
EVTargetCurrent()
EVMaximumVoltageLimit()
EVMaximumCurrentLimit()
EVMaximumPowerLimit()
BulkChargingComplete()
ChargingComplete()
RemainingTimeToFullSoC()
RemainingTimeToBulkSoC()
EVTargetVoltage()
CurrentDemandReq()
CurrentDemandRes()
EVSEIsolationStatus()
EVSEStatusCode()
NotificationMaxDelay()
EVSENotification()
EVSEPresentVoltage()
EVSEPresentCurrent()
EVSECurrentLimitAchieved()
EVSEVoltageLimitAchieved()
EVSEPowerLimitAchieved()


EVSEMaximumVoltageLimit()
EVSEMaximumCurrentLimit()
EVSEMaximumPowerLimit()
ResponseCode()
ChargingControl(StartWeldingDetection)
StateMachineStatus(PowerDelivery)
ChargingProfilePtr()
EVPowerDeliveryParameterFlag()
EVReady()
EVCabinConditioning()
EVRESSConditioning()
EVErrorCode()
EVRESSSOC()
BulkChargingComplete()
ChargingComplete()
PowerDeliveryReq()
PowerDeliveryRes()
EVSEIsolationStatus()
EVSEStatusCode()
NotificationMaxDelay()
EVSENotification()
ResponseCode()
WaitForStateB()
ControlPilotState(State_B)
StateMachineStatus(WeldingDetection)
loop WeldingDetection[ChargingControl!=StopCharging]
EVReady()
EVCabinConditioning()
EVRESSConditioning()
EVErrorCode()
EVRESSSOC()
WeldingDetectionReq()
WeldingDetectionRes()
EVSEIsolationStatus()
EVSEStatusCode()
NotificationMaxDelay()
EVSENotification()
EVSEPresentVoltage()
ResponseCode()

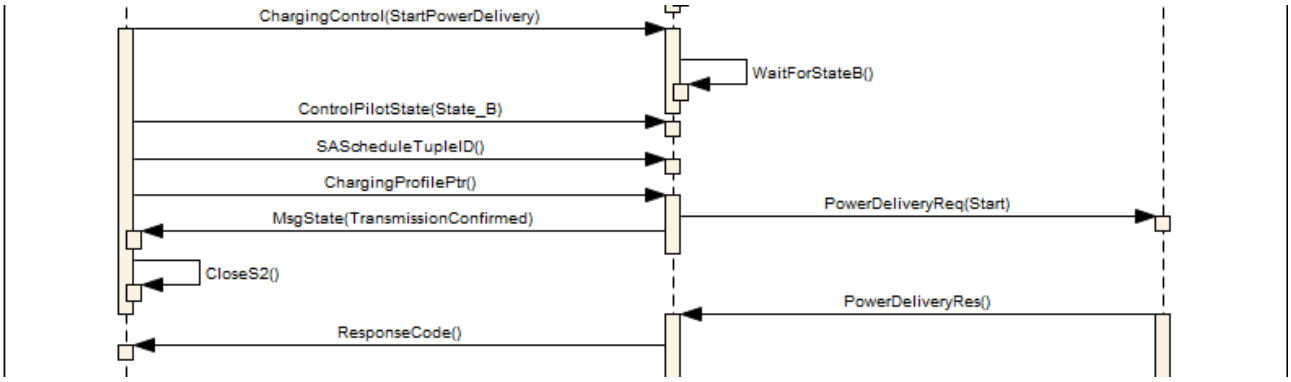


4.5 Encryption of Private Keys

SCC offers the possibility to encrypt the Private Keys of the Contract and Provisioning Certificate in the NVM. The Private Key is encrypted using AES128, which is also used in the ISO15118 specification [1]. The key that is used during encryption is generated by the Password-based Key Derivation Function 2, in short PBKDF2 [3]. It uses a Password and an optional Salt of variable length to create the key. The used algorithm is HMAC_SHA1, which for example is also used for WPA2. The number of iterations is configurable in the configuration tool. The Password and the Salt are provided through the callback API. In case the Password and/or Salt shall be changed, without updating the Private Key, the Scc_ChangeContrCertPrivKeyPassword() or Scc_ChangeProvCertPrivKeyPassword() API may be used.

4.6 Handling of S2 during AC charging

As specified in ISO/IEC 15118-2 [1] with the requirements V2G2-847 and V2G2-848, the S2 has to change at least 250ms after sending the PowerDeliveryReq message. To allow the application of the customer to change the S2 after the transmission of the PowerDeliveryReq, but before the 250ms timeout is elapsed, the following approach is recommended:



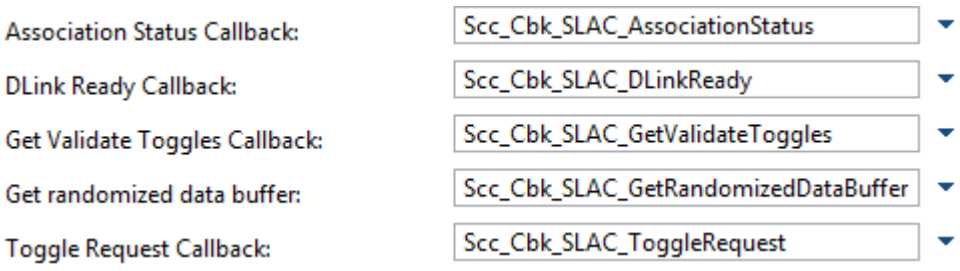
After the application signaled that the module shall continue with the PowerDeliveryReq, the module will check if the Control Pilot is set to State B, which means that S2 is opened. [In case there is a nominal duty cycle and Basic Charging is active, this check will be omitted.] If State B is detected, the module will receive the SAScheduleTupleID and ChargingProfilePtr from the application, before it then finally sends out the PowerDeliveryReq. If TCP confirmed that the SECC has received the PowerDeliveryReq,

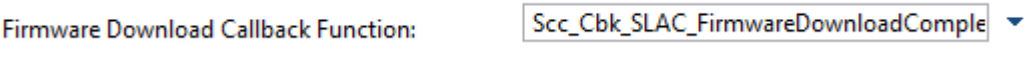
the module will notify the application by calling MsgState(TransmissionConfirmed). The application of the customer now has 250ms to close the S2. After the S2 is closed, the SECC should respond with the PowerDeliveryRes.

This behavior can also be applied when the charging shall be stopped and the S2 has therefore to be opened between the PowerDeliveryReq and PowerDeliveryRes.

4.7 Handling of SLAC by SCC

The SCC is able to handle the SLAC process instead of the application. This way the application only needs to control one lower layer component. To enable SLAC handling by SCC, there are a few things that need to be configured:

- "Enable SLAC Handling" in the configuration tool.
  - Scc -> SccGeneral -> SccV2GConfig -> SccEnableSlacHandling
- Configure the necessary SLAC callbacks
  - EthTrcv -> EthTrcvConfigSet -> EthTrcvSlacConfig -> EthTrcvSlacCallbacks
  - 

- Configure the firmware download complete callback
  - EthTrcv -> EthTrcvGeneral -> EthTrcvFirmwareDownloadConfigSet -> EthTrcvFwStartCbkFct
  - 

- Configure the SLAC callbacks in the SCC Parameter Mapping
  - The Start Mode is according to the EthTrcv technical reference.
  - The QCA idle timer defines the time between the completion of the firmware download and the trigger to start SLAC. A delay is necessary since the QCA is not responding for some time after the firmware download is complete. According to current experiences, this time is between 3 and 5 seconds. But an exact time cannot be provided.

5 API Description

5.1 Type Definitions

The types defined by the **SCC** are described in this chapter. All types and enumerations are prefixed with "Scc_" which has been omitted here for better readability.

Type Name	Value Range
MsgTrigType	`MsgTrig_Message` Request of triggered message will be sent once preconditions have been met.
MsgStatusType	`MsgStatus_Message_OK` The response of the triggered message was received from the EVSE.
	`MsgStatus_Message_Failed` An error occurred during either transmitting the request or receiving the response of the triggered message.
MsgStateType	`MsgState_WaitForNextRequest` The module is currently waiting for the application to trigger the next message.
	`MsgState_StreamingRequest` The module is currently Tx streaming the current request message.
	`MsgState_RequestSent` The triggered request was sent, waiting for the response.
	`MsgState_TransmissionConfirmed` The transmission of the request was confirmed by the transport layer.
	`MsgState_ResponseReceived` The response message was received and is currently being processed.
	`MsgState_Timeout` The response message was not received within the timeout.
StackErrorType	`StackError_NoError` No error occurred yet.
	`StackError_TransportLayer` The transport layer connection was disrupted.
	`StackError_InvalidTxParameter` The application provided an invalid parameter to the module.
	`StackError_Timeout` A timeout occurred.
	`StackError_IpBase` An IpBase API returned a negative value.
	`StackError_Exi` An Exi API returned a negative value.
	`StackError_XmlSecurity` An XmlSecurity API returned a negative value.
	`StackError_Crypto` A Crypto API returned a negative value.

	`StackError_InvalidRxMessage` The message received from the EVSE is invalid.
	`StackError_InvalidRxParameter` The message received from the EVSE contained an invalid parameter.
	`StackError_NegativeResponseCode` The ResponseCode in the response message was negative.
	`StackError_NvM` An error occurred during an NvM operation.
SDPSecurityType	`SDPSecurity_Tls` TLS will be used for the transport layer connection.
	`SDPSecurity_None` Only TCP will be used for the transport layer connection.
SAPSchemaIDType	`SAPSchemaIDs_ISO_FDIS` Schema according to ISO15118 FDIS will be used.
	`SAPSchemaIDs_DIN` Schema according to DIN70121 RC6 will be used.
ChargingControlType	`ChargingControl_None` Nothing to do.
	`ChargingControl_NegotiateChargingParameters` Module will continue by sending the ChargeParameterDiscoveryReq.
	`ChargingControl_PreChargeCompleted` Module will break from the PreCharge loop.
	`ChargingControl_StartPowerDelivery` Module will continue by sending the PowerDeliveryReq.
	`ChargingControl_StartWeldingDetection` Module will stop charging, by entering the WeldingDetection loop.
	`ChargingControl_Renegotiation` Module will start a renegotiation.
	`ChargingControl_StopCharging` Module will stop charging.
FunctionControlType	`FunctionControl_None` Nothing to do.
	`FunctionControl_ChargingMode` Module will start the V2G communication session.
	`FunctionControl_DiagMode` Module will enter the diagnostic mode. Anything related to installed certificates can only be read/written while module is in diagnostic mode.
	`FunctionControl_Reset` Module will be reset.
ControlPilotStateType	`ControlPilotState_None` The state of the ControlPilot is not available yet.
	`ControlPilotState_State_A_E_F` The cable is either plugged out, or an error occurred.

	ControlPilotState_State_B The cable is plugged in, but S2 was not closed yet.
	ControlPilotState_State_C_D The cable is plugged in and S2 is closed.
TCPSocketStateType	TCPSocketState_Closed TCP is not connected to the SECC; the socket is closed and can be used.
	TCPSocketState_Connecting TCP is currently connecting to the SECC.
	TCPSocketState_Connected TCP is connected to the SECC.
	TCPSocketState_Disconnecting TCP is currently disconnecting from the SECC.
PWMStateType	PWMState_ChargingNotAllowed Charging is not allowed, since there is neither a 5% nor a nominal duty cycle.
	PWMState_HLCOptional A nominal duty cycle was detected.
	PWMState_HLCOptional_BCActive A nominal duty cycle was detected and Basic Charging was started.
	PWMState_HLCOnly A 5% duty cycle was detected.
StateM_MsgStateType	SMMS_PreparingRequest Waiting for all required data, before sending next request.
	SMMS_WaitingForResponse Request has been sent, waiting for the response from the EVSE.
	SMMS_ProcessingResponse Received the response, processing the content.
	SMMS_WaitingForApplication Waiting for application to allow to continue with next request.
StateM_StateMachineErrorType	SMER_NoError No error, State Machine is working normally.
	SMER_Timer_<Timer> The <Timer> elapsed after the configured timeout.
	SMER_Initialization_ContractSelectionFailed Invalid contract was selected, check configuration parameter.
	SMER_SECCDiscoveryProtocol_NoSharedSecurityOption The EV and EVSE share no common Security option.
	SMER_ServiceDiscovery_InvalidEnergyTransferMode The EVSE did not offer a valid Energy Transfer Mode.
	SMER_ServiceDiscovery_NoSharedPaymentOption The EV and EVSE share no common Payment Option.
	SMER_ServiceDiscovery_NoContractAndNoCertificateService… There is no valid contract nor Certificate Installation available..

	SMER_ServiceDetail_NoContractAndNoCertificateService… There is no valid contract nor Certificate Installation available..
	SMER_StackError A Stack Error occurred. Please check the StackError signal.
StateM_StateMachineStatusType	StateMachineStatus_None The State Machine was not initialized yet.
	StateMachineStatus_Initialized The State Machine is initialized.
	StateMachineStatus_SLAC SLAC is currently being executed.
	StateMachineStatus_WaitForIP The State Machine waits for an IPv6 address to be assigned.
	StateMachineStatus_SECCDiscoveryProtocol SECC Discovery Protocol is being executed.
	StateMachineStatus_TLConnection The transport layer connection is being established. Either via TCP or TLS.
	StateMachineStatus_SupportedAppProtocol The Supported App Protocol is currently determining the schema that shall be used during the upcoming V2G Communication Session.
	StateMachineStatus_<V2GMessage> The reported V2G Message is currently being handled.
	StateMachineStatus_StopCommunicationSession The V2G Communication Session was stopped successfully; the transport layer connection will therefore now be terminated.
	StateMachineStatus_Finished The State Machine has finished charging and is currently idle.
	StateMachineStatus_Error_WaitingForRetry An error occurred; the State Machine will automatically try again.
	StateMachineStatus_Error_Stopped An error occurred; the State Machine is therefore currently idle.
DynConfigParamsType	Scc_DynConfigParam_<ConfigurationParameter> Contains all configuration parameters that were set to dynamic.
DiagParamsType	DP_ContractCertificate [uint8*]
	DP_ContractCertificateChainSize [uint8*]
	DP_EMAID [uint8*]
	DP_ContractCertificatePrivateKey [uint8*]
	DP_ProvisioningCertificate [uint8*]
	DP_ProvisioningCertificatePrivateKey [uint8*]
	DP_ContractSubCertificate1+x [uint8*] This enum is used to access ContractSubCertificates #(x+1).
	DP_RootCertificate1+x [uint8*] This enum is used to access RootCertificate #(x+1).

Table 7 Type definitions

5.2 Services provided by SCC

The **SCC** API consists of services, which are realized by function calls.

5.2.1 Scc_GetVersionInfo

Prototype	
`void` **`Scc_GetVersionInfo`** `(Std_VersionInfoType *VersionInfoPtr)`	
Parameter	
VersionInfoPtr [in]	pointer for version info
Return code	
void	none
Functional Description	
Scc_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
Call context	
> initialization or task level	

Table 5-8 Scc_GetVersionInfo

5.2.2 Scc_InitMemory

Prototype	
`void` **`Scc_InitMemory`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
initializes global variables	
Particularities and Limitations	
AUTOSAR extension has to be called before any other calls to the module	
Call context	
> task level	

Table 5-9 Scc_InitMemory

5.2.3 Scc_Init

Prototype	
`void Scc_Init (Scc_ConfigType *CfgPtr)`	
Parameter	
CfgPtr [in]	pointer to module configuration
Return code	
void	none
Functional Description	
stores the start address of the post build time configuration of the module and may be used to initialize the data structures	
Particularities and Limitations	
has to be called before useage of the module	
Call context	
> initialization	

Table 5-10 Scc_Init

5.2.4 Scc_MainFunction

Prototype	
`void Scc_MainFunction (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
re-initialize closed connections and transmit pending data	
Particularities and Limitations	
Call context	
> task level	

Table 5-11 Scc_MainFunction

5.2.5 Scc_ResetSessionID

Prototype
`Scc_ReturnType Scc_ResetSessionID (boolean ForceReset)`

Parameter	
ForceReset [in]	force the reset, even when a V2G Communication Session is ongoing
Return code	
Scc_ReturnType	E_OK SessionID has been reset
Scc_ReturnType	E_NOT_OK a V2G session is currently active, please try again later
Functional Description	
resets the SessionID to '0x00'	
Particularities and Limitations	
will only have effect outside of a V2G session	
Call context	
> task level	

Table 5-12 Scc_ResetSessionID

5.2.6 Scc_DynConfigDataReadAccess

Prototype
`Scc_ReturnType` **`Scc_DynConfigDataReadAccess`** `(Scc_DynConfigParamsType DataID, uint16 *DataPtr)`
Parameter
DataID [in] — data identifier
DataPtr [out] — pointer for diagnostic data
Return code
Scc_ReturnType — E_OK configuration data was successfully read
Scc_ReturnType — E_NOT_OK invalid DataID
Functional Description
configuration data read access
Particularities and Limitations
Call context
> task level

Table 5-13 Scc_DynConfigDataReadAccess

5.2.7 Scc_DynConfigDataWriteAccess

Prototype
`Scc_ReturnType` **`Scc_DynConfigDataWriteAccess`** `(Scc_DynConfigParamsType DataID, uint16 Data)`

Parameter	
DataID [in]	data identifier
Data [in]	configuration data that shall be written to NVRAM
Return code	
Scc_ReturnType	E_OK configuration data was successfully written
Scc_ReturnType	E_NOT_OK invalid DataID
Functional Description	
configuration data write access	
Particularities and Limitations	
Call context	
> task level	

Table 5-14 Scc_DynConfigDataWriteAccess

5.2.8 Scc_DiagDataReadAccess

Prototype	
`Scc_ReturnType` **`Scc_DiagDataReadAccess`** `(Scc_DiagParamsType DataID, uint8 *DataPtr, uint16 *DataLenPtr)`	
Parameter	
DataID [in]	data identifier
DataPtr [out]	pointer for diagnostic data
DataLenPtr [out]	pointer for maximum / actual length of diagnostic data in bytes
Return code	
Scc_ReturnType	OK diagnostic data was successfully read
	Pending the requested data is currently being read from NVRAM
	NotOK an error occurred
Functional Description	
diagnostic data read access	
Particularities and Limitations	
Call context	
> task level	

Table 5-15 Scc_DiagDataReadAccess

5.2.9 Scc_DiagDataWriteAccess

Prototype	
`Scc_ReturnType` **`Scc_DiagDataWriteAccess`** `(Scc_DiagParamsType DataID, uint8 *DataPtr, uint16 DataLen)`	
Parameter	
DataID [in]	data identifier
DataPtr [in]	pointer with address of the diagnostic data
DataLen [in]	length of the diagnostic data in bytes
Return code	
Scc_ReturnType	OK diagnostic data written
	NotOK invalid parameter (data identifier not found, NULL_PTR parameter, invalid length)
	Pending NvM is currently still processing, keep calling this API
	Busy diagnostic job currently not possible, try again later
Functional Description	
diagnostic data write access	
Particularities and Limitations	
Call context	
> task level	

Table 5-16 Scc_DiagDataWriteAccess

5.2.10 Scc_DiagDataGetBlockStatus

Prototype	
`Scc_ReturnType` **`Scc_DiagDataGetBlockStatus`** `(Scc_DiagParamsType DataID, uint8 *NvmResultPtr)`	
Parameter	
DataID [in]	data identifier
NvmResultPtr [out]	current status of the requested block (NvM_RequestResultType)
Return code	
Scc_ReturnType	E_OK operation was successful
Scc_ReturnType	E_NOT_OK invalid DataID or certificate index
Functional Description	
provides information about the current status of the NvM block of the diag data	
Particularities and Limitations	
Call context	

> task level

Table 5-17 Scc_DiagDataGetBlockStatus

5.2.11 Scc_SwapContrCertChain

Prototype	
`Scc_ReturnType` **`Scc_SwapContrCertChain`** `(uint8 NewContractCertificateChainIndex, boolean ForceSwap)`	
Parameter	
NewContractCertificateChainIndex [in]	new slot index
ForceSwap [in]	force to swap the chain (i.e. reset the current chain status)
Return code	
Scc_ReturnType	OK chain swapped successfully
	Busy a V2G session is active, swap is currently not possible
	NotOK invalid chain index
Functional Description	
swap the used contract certificate chain slot	
Particularities and Limitations	
not available during an active V2G session, unless ForceSwap is used	
Call context	
> task level	

Table 5-18 Scc_SwapContrCertChain

5.2.12 Scc_CheckContrCertValidity

Prototype	
`Scc_ReturnType` **`Scc_CheckContrCertValidity`** `(uint32 DateTimeNow, uint32 DateTimeThreshold, Scc_ContrCertStatusType *ContrCertStatus)`	
Parameter	
DateTimeNow [in]	current time
DateTimeThreshold [in]	a time in the future to check whether the certificate expires until then
ContrCertStatus [out]	current status of the contract certificate
Return code	
Scc_ReturnType	none
Functional Description	
check if the currently installed certificate is expired or will expire soon	

Particularities and Limitations
the certificates have to be read from the NvM
Call context
> task level

Table 5-19 Scc_CheckContrCertValidity

5.2.13 Scc_ResetNvMBlockStatus

Prototype	
`Scc_ReturnType` **`Scc_ResetNvMBlockStatus`** `(Scc_DiagParamsType NvMBlock)`	
Parameter	
NvMBlock [in]	NvM block of which the status shall be reset
Return code	
Scc_ReturnType	OK status was reset, NvM block can be read again from NVRAM
	Busy a V2G session is currently active, try again later or stop charging
	Pending NvM block is busy, keep calling this API
	NotOK invalid NvMBlock or index (Sub or Root Certificates) selected
Functional Description	
reset the status of a RAM block of an NvM block, so it can be read again	
Particularities and Limitations	
will only have effect outside of a V2G session	
Call context	
> task level	

Table 5-20 Scc_ResetNvMBlockStatus

5.2.14 Scc_DeleteContract

Prototype	
`Scc_ReturnType` **`Scc_DeleteContract`** `(uint8 ContractIdx)`	
Parameter	
ContractIdx [in]	the slot of the contract certificate that shall be deleted, starts with '0'
Return code	
Scc_ReturnType	OK contract deleted successfully
	Busy a V2G session is currently active, wait until it is finished or stop charging
	NotOK an error occurred
Functional Description	
deletes the Contract Certificate including all sub certificates and the private key	

Particularities and Limitations
not available during an active V2G session
Call context
> task level

Table 5-21 Scc_DeleteContract

5.2.15 Scc_DeleteRootCert

Prototype	
`Scc_ReturnType` **`Scc_DeleteRootCert`** `(uint8 RootCertIdx)`	
Parameter	
RootCertIdx [in]	the slot of the root certificate that shall be deleted, starts with '0'
Return code	
Scc_ReturnType	OK root certificate deleted successfully
	Busy a V2G session is currently active, wait until it is finished or stop charging
	NotOK an error occurred
Functional Description	
deletes the selected Root Certificate	
Particularities and Limitations	
not available during an active V2G session	
Call context	
> task level	

Table 5-22 Scc_DeleteRootCert

5.2.16 Scc_ChangeContrCertPrivKeyPassword

Prototype	
`Scc_ReturnType` **`Scc_ChangeContrCertPrivKeyPassword`** `(Scc_BufferPointerType *NewPasswordPtr, Scc_BufferPointerType *NewSaltPtr)`	
Parameter	
NewPasswordPtr [in]	pointer to the new password
NewSaltPtr [in]	pointer to the new salt (optional, if not used set to NULL)
Return code	
Scc_ReturnType	OK key was changed successfully
	Busy change of key currently not possible because of an active V2G session
	Pending change of key ongoing
	NotOK an error occurred while changing the key

Functional Description
changes the key of the stored private key
Particularities and Limitations
not available during an active V2G session
Call context
> task level

Table 5-23 Scc_ChangeContrCertPrivKeyPassword

5.2.17 Scc_ChangeProvCertPrivKeyPassword

Prototype	
`Scc_ReturnType` **`Scc_ChangeProvCertPrivKeyPassword`** `(Scc_BufferPointerType *NewPasswordPtr, Scc_BufferPointerType *NewSaltPtr)`	
Parameter	
NewPasswordPtr [in]	pointer to the new password
NewSaltPtr [in]	pointer to the new salt (optional, if not used set to NULL)
Return code	
Scc_ReturnType	OK key was changed successfully
	Busy change of key currently not possible because of an active V2G session
	Pending change of key ongoing
	NotOK an error occurred while changing the key
Functional Description	
changes the key of the stored private key	
Particularities and Limitations	
not available during an active V2G session	
Call context	
> task level	

Table 5-24 Scc_ChangeProvCertPrivKeyPassword

5.2.18 Scc_ValidateContrCertKeyPair

Prototype	
`Scc_ReturnType` **`Scc_ValidateContrCertKeyPair`** `(void)`	
Parameter	
void	none
Return code	
Scc_ReturnType	OK private and public key pair validated successfully

	Pending contract certificate or its private key are still loaded from NVRAM
	NotOK NvM read error, or public and private key are no pair
Functional Description	
validates the contract certificate by checking the private key against the public key	
Particularities and Limitations	
contract certificate has to be read from NVRAM	
Call context	
> task level	

Table 5-25 Scc_ValidateContrCertKeyPair

5.2.19 Scc_ValidateProvCertKeyPair

Prototype	
`Scc_ReturnType` **`Scc_ValidateProvCertKeyPair`** `(void)`	
Parameter	
void	none
Return code	
Scc_ReturnType	OK private and public key pair validated successfully
	Pending provisioning certificate or its private key are still loaded from NVRAM
	NotOK NvM read error, or public and private key are no pair
Functional Description	
validates the provisioning certificate by checking the private key against the public key	
Particularities and Limitations	
provisioning certificate has to be read from NVRAM	
Call context	
> task level	

Table 5-26 Scc_ValidateProvCertKeyPair

5.2.20 Scc_GetCertDistinguishedNameObject

Prototype	
`Scc_ReturnType` **`Scc_GetCertDistinguishedNameObject`** `(uint8 *CertPtr, uint16 CertLen, uint8 *DataPtr, uint16 *DataLenPtr, Scc_BERObjectIDsType ObjectID)`	
Parameter	
CertPtr [in]	pointer to certificate
CertLen [in]	certificate length in bytes
DataPtr [out]	pointer to output buffer

inout] [out]	DataLenPtr pointer to the length of the output buffer
ObjectID [in]	defines which object will be returned
SearchInIssuer [in]	defines whether the CN of the Issuer or the Subject will be returned
Return code	
Scc_ReturnType	E_OK value retrieved
Scc_ReturnType	E_NOT_OK invalid parameter (value not found, NULL_PTR parameter, invalid length)
Functional Description	
returns the selected object of the DistinguishedName of the provided certificate	
Particularities and Limitations	
The function can be used for every object from the Enum Scc_BERObjectIDsType.	
Call context	
> task level	

Table 5-27 Scc_GetCertDistinguishedNameObject

5.2.21 Scc_GetIndexOfPublicKey

Prototype	
`Scc_ReturnType` **`Scc_GetIndexOfPublicKey`** `(uint8 *CertPtr, uint16 CertLen, uint16 *PubKeyIdxPtr, uint16 *PubKeyLenPtr)`	
Parameter	
CertPtr [in]	pointer to the certificate of which the public key shall be used
CertLen [in]	length of the certificate
PubKeyPtr [out]	pointer to the extracted public key
PubKeyLenPtr [out]	length of the extracted public key
Return code	
Scc_ReturnType	E_OK public key found
Scc_ReturnType	E_NOT_OK public key not found
Functional Description	
returns the index of the public key which is contained in the certificate	
Particularities and Limitations	
Call context	
> task level	

Table 5-28 Scc_GetIndexOfPublicKey

5.2.22 Scc_GetCertFromPKCS7

Prototype	
`Scc_ReturnType Scc_GetCertFromPKCS7 (const uint8 *CertChainMsgPtr, const uint16 *CertChainLengthPtr, const uint8 *NumberOfCerts, Scc_CertificateType *CertChainStructPtr)`	
Parameter	
CertChainMsgPtr [in]	pointer to the PKCS#7 structure
CertChainLengthPtr [in]	pointer to the length of the PKCS#7 structure
NumberOfCerts [in]	number of certificates in the PKCS#7 structure
CertChainStructPtr [out]	pointer the concatenated list of type Scc_CertificateType
Return code	
Scc_ReturnType	E_OK public key found
Scc_ReturnType	E_NOT_OK public key not found
Functional Description	
Gets the certificate positions and length out of a DER decoded PKCS#7 structure.	
Particularities and Limitations	
Call context	
> task level	

Table 5-29 Scc_GetCertFromPKCS7

5.3 Services used by SCC

In the following table services provided by other components, which are used by the **SCC** are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
Crypto	`Crypto_ValidateEcdsaSignature`
Crypto	`esl_initWorkspaceHeader`
Crypto	`esl_initGenerateSharedSecretDHEcP_prim`
Crypto	`esl_initDecryptAES128`
Crypto	`esl_decryptAES128`
Crypto	`esl_initSHA256`
Crypto	`esl_updateSHA256`
Crypto	`esl_finalizeSHA256`
Dem	`Dem_ReportErrorStatus`
Det	`Det_ReportError`
Exi	`Exi_InitEncodeWorkspace`
Exi	`Exi_Encode`
Exi	`Exi_FinalizeExiStream`

Component	API
Exi	Exi_InitDecoderWorkspace
Exi	Exi_Decode
IpBase	IpBase_Copy
IpBase	IpBase_StrCmpLen
IpBase	IpBase_StrCpy
IpBase	IpBase_BerInitWorkspace
IpBase	IpBase_BerGetElement
IpBase	IpBase_ConvString2Int
IpBase	IpBase_CopyString2PbufAt
IpBase	IPBASE_SWAP16
IpBase	IPBASE_SWAP32
Tls	Tls_Connect
Tls	Tls_ProvideTxBuffer
Tls	Tls_TransmitTo
Tls	Tls_Received
Tls	Tls_Close
Tls	Tls_ChangeParameterRequest
XmlSecurity	XmlSecurity_InitSigValWorkspace
XmlSecurity	XmlSecurity_ValidateExiSignature
XmlSecurity	XmlSecurity_InitSigGenWorkspace
XmlSecurity	XmlSecurity_AddExiReference
XmlSecurity	XmlSecurity_GenerateSignature
NvM	NvM_ReadBlock
NvM	NvM_Write
NvM	NvM_InvalidateNvBlock

Table 30 Services used by the **SCC**

5.4 Callback Functions

This chapter describes the callback functions that are implemented by the **SCC** and can be invoked by other modules. The prototypes of the callback functions are provided in the header file Scc_Cbk.h.

5.4.1 Scc_Cbk_TL_RxIndication

Prototype	
`void` **`Scc_Cbk_TL_RxIndication`**`(TcpIp_SocketIdType SockHnd, const TcpIp_SockAddrType SourcePtr, uint8 *DataPtr, uint16 DataLen)`	
Parameter	
SockHnd	socket handle
SourcePtr	source network address and port

DataPtr	pointer to the received data
DataLen	length of the received data
Return code	
void	none
Functional Description	
receive indication	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 31 Scc_Cbk_TL_RxIndication

5.4.2 Scc_Cbk_TL_TCPAccepted

Prototype
`void` **`Scc_Cbk_TL_TCPAccepted`**`(TcpIp_SocketIdType SockHnd, TcpIp_SocketIdType SocketIdConnected, TcpIp_SockAddrType RemoteAddrPtr)`
Parameter
SockHnd: socket handle
SocketIdConnected: socket handle of the newly created socket
RemoteAddrPtr: remote address of the TCP client that connected
Return code
void: none
Functional Description
TCP connection accepted.
Particularities and Limitations
Only relevant for TCP servers.
Call Context
interrupt or task level

Table 32 Scc_Cbk_TL_TCPAccepted

5.4.3 Scc_Cbk_TL_TCPConnected

Prototype
`void` **`Scc_Cbk_TL_TCPConnected`**`(TcpIp_SocketIdType SockHnd)`
Parameter
SockHnd: socket handle
Return code
void: none

Functional Description
TCP connected.
Particularities and Limitations
none
Call Context
interrupt or task level

Table 33 Scc_Cbk_TL_TCPConnected

5.4.4 Scc_Cbk_TL_TCPEvent

Prototype	
`void` **`Scc_Cbk_TL_TCPEvent`**`(TcpIp_SocketIdType SockHnd, IpBase_TcpIpEventType Event)`	
Parameter	
SockHnd	socket handle
Event	socket event
Return code	
void	none
Functional Description	
TCP event handling.	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 34 Scc_Cbk_TL_TCPEvent

5.4.5 Scc_Cbk_TLS_CertChain

Prototype	
`void` **`Scc_Cbk_TLS_CertChain`**`(TcpIp_SocketIdType SockHnd, uint8 *validationResultPtr, uint8 * certChainPtr, uint32 certChainLen)`	
Parameter	
SockHnd	socket handle
validationResultPtr	Validation result (VALIDATION_OK, VALIDATION_UNKNOWN_CA)
certChainPtr	pointer to start of the certificate chain
certChainLen	length of the certificate chain
Return code	
void	none
Functional Description	
TLS validation result for a server certificate chain	

Particularities and Limitations
none
Call Context
interrupt or task level

Table 35 Scc_Cbk_TLS_CertChain

5.4.6 Scc_Cbk_Eth_TransceiverLinkStateChange

Prototype	
`void` **`Scc_Cbk_Eth_TransceiverLinkStateChange`**`(uint8 CtrlIdx, EthTrcv_LinkStateType TrcvLinkState)`	
Parameter	
CtrlIdx	index of the Ethernet controller
TrcvLinkState	new state of the tranceiver
Return code	
void	none
Functional Description	
Called by lower layer (e.g. EthIf) to indicate a change of the transceiver link state	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 36 Scc_Cbk_Eth_TransceiverLinkStateChange

5.4.7 Scc_Cbk_IP_AddressAssignmentChange

Prototype	
`void` **`Scc_Cbk_IP_AddressAssignmentChange`**`(uint8 CtrlIdx, boolean Assigned)`	
Parameter	
CtrlIdx	controller index
Assigned	assignment flag
Return code	
void	none
Functional Description	
IP address assignment change callback.	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 37 Scc_Cbk_IP_AddressAssignmentChange

5.4.8 Scc_Cbk_SLAC_FirmwareDownloadComplete

Prototype	
`void` **`Scc_Cbk_SLAC_FirmwareDownloadComplete`**`(void)`	
Parameter	
void	
Return code	
void	
Functional Description	
callback that is called after the firmware download was processed and the firmware was started	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 38 Scc_Cbk_SLAC_FirmwareDownloadComplete

5.4.9 Scc_Cbk_SLAC_AssociationStatus

Prototype	
`void` **`Scc_Cbk_SLAC_AssociationStatus`**`(uint8 AssociationStatus)`	
Parameter	
AssociationStatus	the association status reported to the application
Return code	
void	none
Functional Description	
callback function for ongoing association errors and status information	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 39 Scc_Cbk_SLAC_AssociationStatus

5.4.10 Scc_Cbk_SLAC_DLinkReady

Prototype	
`void` **`Scc_Cbk_SLAC_DLinkReady`**`(EthTrcv_LinkStateType DLinkReady, uint8* NMKPtr, uint8* NIDPtr)`	
Parameter	
DLinkReady	the power line link state after SLAC was finished
NMKPtr	the network membership key (NMK) that was established during the SLAC session
NIDPtr	the network identifier (NID) that was established during the SLAC session

Return code	
void	none
Functional Description	
D-LINK READY indication informs higher layers about a change of PLC link status	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 40 Scc_Cbk_SLAC_DLinkReady

5.4.11 Scc_Cbk_SLAC_GetRandomizedDataBuffer

Prototype	
`void Scc_Cbk_SLAC_GetRandomizedDataBuffer(uint8* RandomDataPtr, uint16 RandomDataLen)`	
Parameter	
RandomDataPtr	the buffer that must be randomized
RandomDataLen	the length of the buffer
Return code	
void	none
Functional Description	
the callback is used by the SLAC component to gain a random byte array	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 41 Scc_Cbk_SLAC_GetRandomizedDataBuffer

5.4.12 Scc_Cbk_SLAC_GetValidateToggles

Prototype	
`uint8 Scc_Cbk_SLAC_GetValidateToggles(void)`	
Parameter	
void	
Return code	
uint8	random number of validation toggles (between 1 and 3)
Functional Description	
callback function to generate a random number of validation toggles	
Particularities and Limitations	
none	

Call Context
interrupt or task level

Table 42 Scc_Cbk_SLAC_GetValidateToggles

5.4.13 Scc_Cbk_SLAC_ToggleRequest

Prototype	
`void` **`Scc_Cbk_SLAC_ToggleRequest`**`(uint8 ToggleNum)`	
Parameter	
ToggleNum	number of BCB-Toggles the application must generate on the Control Pilot
Return code	
void	
Functional Description	
callback to request the toggle process during the validation phase	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 43 Scc_Cbk_SLAC_ToggleRequest

5.5 Parameter Callback Interface

This component provides a parameter callback interface which is used for the communication with the application. The name of the function which has to be provided by the application must be set in the configurator. Otherwise a default function name will be generated based on the template Appl_SccCbk_[Set|Get]_<Module>_<ParameterName>.

For a description of the ISO15118 relevant parameters please refer to the specification itself. All other parameters are described in the following table:

5.5.1 Rx Parameter

Parameter Name	C-Type	Description	Value Range
Core			
CyclicMsgTrigRx	boolean	Required for cyclic messages	`true` Another cyclic message such as AuthorizationReq will be sent.
MsgTrig	enum	Message trigger	`Scc_MsgTrigType`
StateM			
ChargingControl	enum	Charging control	`Scc_StateM_ChargingControlType`
ControlPilotState	enum	Current state of the ControlPilot	`Scc_StateM_ControlPilotStateType`
EnergyTransferMode	enum	Chosen energy transfer mode	`Scc_StateM_EnergyTransferModeType`

EVTimeStamp	uint32	UNIX timestamp	`unsigned 32 bit`
FunctionControl	enum	Module control	`Scc_StateM_ FunctionControlType`
PWMState	enum	Current PWM state.	`Scc_StateM_ PWMStateType`
Slac			
StartMode	enum	SLAC start mode	`EthTrcv_30_Ar7000_ Slac_StartModeType`
QCAIdleTimer	uint16	QCA idle time	`unsigned 16 bit` MainFunction cycles

Table 44 Rx Callback Parameters

5.5.2 Tx Parameter

Parameter Name	C-Type	Description	Value Range
Core			
CyclicMsgRcvd	boolean	Is called when a cyclic message was received.	`true`
CyclicMsgTrigTx	boolean	Resets the trigger at the application when the next request was sent.	`false`
IPAssigned	boolean	Informs the application whether an IP address was assigned to the PLC stack or not.	`true` An IP address was assigned.
			`false` The IP address was lost.
MsgState	enum	Reports the processing state of the current message.	`Scc_MsgStateType`
MsgStatus	enum	Reports the status of the triggered message.	`Scc_MsgStatusType`
SAPSchemaID	enum	Informs which schema is used for this session.	`Scc_SAPSchemaIDType`
StackError	enum	Detailed information about message status.	`Scc_StackErrorType`
StateM			
InternetAvailable	boolean	Status of VAS internet connection.	`true` Internet connection is available.
StateMachineError	enum	Provides information about the error that occurred.	`Scc_StateM_ StateMachineErrorType`
StateMachineStatus	enum	The current status of the state machine.	`Scc_StateM_ StateMachineStatusType`
EnergyTransferModeFlags	uint8 (enum)	Provides information about the available EnergyTransferModes. Encoded as bit flags!	`Scc_StateM_ EnergyTransferModeType`
Slac			

AssociationStatus	uint8	Association Status of SLAC	See technical reference of EthTrcv for details
NMK	struct*	Network Membership Key (struct pointer is only valid during the callback)	`Scc_BufferPointerType*`
NID	struct*	Network Identifier (struct pointer is only valid during the callback)	`Scc_BufferPointerType*`
ToggleRequest	uint8	Number of BCB toggles the application must generate	`1-3`

Table 45 Tx Callback Parameters

6 Configuration

In **SCC** the attributes can be configured according to/ with the following methods / tools:

> Configuration in DaVinci Configurator 5

For a detailed description see the comments for each parameter in DaVinci Configurator 5.

6.1 Configuration Variants

The **SCC** currently only supports the configuration variant

`VARIANT-PRE-COMPILE`

6.2 Maximum sizes of EXI Streams and EXI Structs for ISO15118 use case

To give an orientation about how big the buffers should be for the ISO15118 use case, the following table will provide an overview about the size of the different message's EXI stream and EXI struct size. These sizes were measured using the Visual Studio 10 compiler without optimizations:

V2G Message	Use Case							
	PNC				EIM			
	AC		DC		AC		DC	
	Stream	Struct	Stream	Struct	Stream	Struct	Stream	Struct
SessionSetupReq	22	58	22	58	22	58	22	58
SessionSetupRes	66	110	66	110	66	110	66	110
ServiceDiscoveryReq	21	124	21	124	21	214	21	124
ServiceDiscoveryRes	335	494	335	494	334	494	334	494
ServiceDetailReq	23	48	23	48	23	48	23	48
ServiceDetailRes	50	204	50	204	50	204	50	204
PaymentServiceSelectionReq	27	54	27	54	24	54	24	54
PaymentServiceSelectionRes	22	50	22	50	22	50	22	50
CertificateInstallationReq	901	74	901	74				
CertificateInstallationRes	3557	6350	3557	6350				
CertificateUpdateReq	1847	98	1847	98				
CertificateUpdateRes	3558	6358	3558	6358				
PaymentDetailsReq	1424	74	1424	74				
PaymentDetailsRes	45	88	45	88				
AuthorizationReq	319	70	319	70	21	58	21	58
AuthorizationRes	23	54	23	54	23	54	23	54
ChargeParameterDiscoveryReq	40	134	51	166	40	134	51	166
ChargeParameterDiscoveryRes	2483	12914	2500	12998	953	4562	970	4646

CableCeckReq			24	58			24	58
CableCeckRes			27	78			27	78
PreChargeReq			31	90			31	90
PreChargeRes			30	90			30	90
PowerDeliveryReq	202	70	206	90	202	70	206	90
PowerDeliveryRes	25	70	26	78	25	70	26	78
ChargingStatusReq	21	46			21	46		
ChargingStatusRes	132	270			39	138		
CurrentDemandReq			50	178			50	178
CurrentDemandRes			64	208			57	314
MeteringReceiptReq	406	88	319	88				
MeteringReceiptRes	25	70	26	78				
WeldingDetectionReq			24	58			24	58
WeldingDetectionRes			30	90			30	90
SessionStopReq	22	50	22	50	22	50	22	50
SessionStopRes	22	50	22	50	22	50	22	50

Table 46 Maximum message sizes

6.3 Configuration parameters

It is now possible to select where the configuration parameters will be stored. There are three options:

1. Static
   - A define will be generated, configuration parameter will thus be stored in ROM.
2. Dynamic
   - An NvM block will be created, in which the configuration parameter will be stored.
   - The value that is specified under "Default Value" will be used to create a ROM default.
3. Extern
   - A callback has to be specified that will be called each time the parameter is required. The callback should return the configuration parameter as a uint16.
   - The uint16 value that is returned for timing parameters needs to have a unit of MainFunctionCycles.
   - Example: 2s Timeout and 20ms MainFunctionCycle lead to a returned value of 100.

Following a list of all available configuration parameters (depending on enabled features):

Name	Unit	Description
Timer General		

Name	Unit	Description
IP Address Wait Timeout	ms	Defines how long the component will wait for an IPv6 address after the application commanded to start the charging process.
SECC Discovery Protocol Retries	#	Defines how many SECCDiscoveryProtocol requests will be sent, if no response is received within the timeout.
SECC Discovery Protocol Timeout	ms	Defines after which time another SECCDiscoveryProtocol request is sent, if no response was received.
Supported App Protocol Message Timeout	ms	Defines the message timeout for the SupportedAppProtocol.
Timer ISO		
Communication Setup Timeout	ms	Defines the timeout for the CommunicationSetup.
<V2GMessage> Message Timeout	ms	Defines the <V2GMessage> message timeout.
Timer ISO		
Communication Setup Timeout	ms	Defines the timeout for the CommunicationSetup.
Ready To Charge Timeout	ms	Defines the timeout for ReadyToCharge.
<V2GMessage> Message Timeout	ms	Defines the <V2GMessage> message timeout.
State Machine		
Accept Unsecure Connection	bool	Defines whether the EV is allowed to charge without TLS.
Authorization Next Request Delay	ms	Defines the time in main function cycles which the EV waits after receiving a AuthorizationRes with EVSEProcessing set to "Ongoing" before sending the next AuthorizationReq.
Authorization Cycle Timeout	ms	Defines the time in seconds which the EV waits for the EVSE to send a AuthorizationRes with EVSEProcessing set to "Finished" after receiving the first AuthorizationRes with EVSEProcessing set to "Ongoing".
Cable Check Next Request Delay	ms	Defines the time in main function cycles which the EV waits after receiving a CableCheckRes with EVSEProcessing set to "Ongoing" before sending the next CableCheckReq.
Cable Check Cycle Timeout	ms	Defines the time in seconds which the EV waits for the EVSE to send a CableCheckRes with EVSEProcessing set to "Finished" after receiving the first CableCheckRes with EVSEProcessing set to "Ongoing".
Certificate Expiration Threshold	ms	If the remaining validity of the contract certificate is shorter than the days specified here, the SCC will try to update it via CertificateUpdate.
Charge Parameter Discovery Next Request Delay	ms	Defines the time in main function cycles which the EV waits after receiving a ChargeParameterDiscoveryRes with EVSEProcessing set to "Ongoing" before sending the next ChargeParameterDiscoveryReq.
Charge Parameter Discovery Cycle Timeout	ms	Defines the time in seconds which the EV waits for the EVSE to send a ChargeParameterDiscoveryRes with EVSEProcessing set to "Finished" after receiving the first ChargeParameterDiscoveryRes with EVSEProcessing set to "Ongoing".

Name	Unit	Description
Charging Status Next Request Delay	ms	Defines the time in main function cycles which the EV waits after receiving a ChargingStatusRes with EVSEProcessing set to "Ongoing" before sending the next ChargingStatusReq.
Contract Certificate Chain Index In Use	#	Defines which Contract Certificate Chain shall be used for the upcoming V2G Communication Session.
Current Demand Next Request Delay	ms	Defines the time in main function cycles which the EV waits after receiving a CurrentDemandRes with EVSEProcessing set to "Ongoing" before sending the next CurrentDemandReq.
Payment Prioritization	enum	Defines which payment option the component shall prefer. Enumeration: EIM_ONLY (0), PNC_ONLY (1), PRIORITIZE_PNC (2)
Pre Charge Next Request Delay	ms	Defines the time in main function cycles which the EV waits after receiving a PreChargeRes before sending the next PreChargeReq.
Pre Charge Cycle Timeout	ms	Defines the time in seconds which the module waits for the EV application to continue with PowerDeliveryReq after having received the first PreChargeRes.
QCA Idle Timer	ms	Defines how long the component will wait for the QCA to be ready before starting SLAC.
Reconnection Delay	ms	Defines the seconds the EV waits before starting a new connection attempt after a previous error.
Reconnection Retries	#	Defines how often the EV will try to reconnect to the EVSE after an error occurred.
Request Certificate Details	bool	Defines whether the EV shall try to request service details for the certificate service of the EVSE.
Request Certificate Installation	bool	Defines whether the EV shall try to request service details for the certificate installation service of the EVSE.
Request Certificate Update	bool	Defines whether the EV shall try to request service details for the certificate update service of the EVSE.
Request Internet Details	bool	Defines whether the EV shall try to request service details for the internet service of the EVSE.
SLAC Start Mode	enum	Sets the start mode of the SLAC module. Use the following enumeration: EthTrcv_30_Ar7000_Slac_StartModeType
Welding Detection Next Request Delay	ms	Defines the time in main function cycles which the EV waits after receiving a WeldingDetectionRes with EVSEProcessing set to "Ongoing" before sending the next WeldingDetectionReq.

Table 47 Configuration Parameters

6.4 ReadAll() & WriteAll() NvM blocks

The following NvM blocks have to be read via NvM_ReadAll() and written via NvM_WriteAll(). The NvM will be informed via the NvM_SetRamBlockStatus() API that there has been a change in the RAM block, if this is the case. The provided ROM Defaults can be used, but can also be changed in case a different configuration is preferred:

NvM Block Name	Variable Name	ROM Default Name
SessionID	Scc_SessionIDNvm	Scc_SessionIDNvmRomDefault
Dynamic Configuration	Scc_DynConfigParamsNvm	Scc_DynConfigParamsNvmRomDefault

NvM Block Name	Variable Name	ROM Default Name
Parameters		

Table 48 NvM Blocks for NvM_ReadAll() & NvM_WriteAll()

6.5 Miscellaneous

6.5.1 Timer based on the MainFunctionCycle

When changing the MainFunction cycle of this module, please keep in mind to also change all diagnostic parameters (such as Authoriztation Next Request Delay) that are based on the MainFunction cycles. Otherwise the timing behavior of the module will deviate.

6.5.2 TcpIp TCP Idle Timeout

Please make sure to increase the TCP Idle Timeout to at least 61 seconds. The default value is 30 seconds. Since the Sequence Performance Timeout has a value of 60 seconds, the TcpIp would otherwise always close the TCP connection before the Sequence Performance Timer could elapse.

The timeout can be found in the GENy configuration here:
TcpIp -> TcpIpConfigSet -> TcpIpTcpGeneral -> Idle Timeout [s]

6.5.3 TcpIp MSL Timeout

The MSL Timeout is used to delay the establishment of a new connection, after the old one was closed. The reason for this is that if a connection goes over the internet, the remote node may still have sent some data, which could take some time. Since in the use case of V2G the nodes are connected directly, this delay is not necessary. The MSL Timeout can therefore be set to 0 seconds.

The timeout can be found in the GENy configuration here:
TcpIp -> TcpIpConfigSet -> TcpIpTcpGeneral -> Msl Timeout [s]

6.5.4 TcpIp Retransmission Timer

The Retransmission Timer is used to retransmit TCP packets, in case they were not acknowledged by the remote node. The default value for this timer is currently set to 2 seconds. Since most of the V2G message timeouts are also set to 2 seconds, they will be already expired in case a request message was not received by the remote node. Therefore it is recommended to reduce the minimum amount of this timer, to e.g. 500ms or even 100ms to allow CurrentDemandReq retransmissions.

The timeout can be found in the DaVinci Configurator Pro configuration here:
TcpIp -> TcpIpConfigSet -> TcpIpTcpGeneral -> Min Tx Retry Interval Time [s]

7 AUTOSAR Standard Compliance

Currently, the component is not considered in AUTOSAR. However, the component is designed based on AUTOSAR principles.

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
Cfg5	Generation tool for MICROSAR components
SysService_SswScc	Vector Informatik component name of the module

Table 49 Glossary

8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
EVCC	Electric Vehicle Communication Controller
EXI	Efficient XML Interchange
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (Vector AUTOSAR solution)
RTE	Runtime Environment
S2	Relay, as specified in IEC 61851-1
SAP	Supported Application Protocol
SCC	Smart Charge Communication
SDP	SECC Discovery Protocol
SECC	Supply Equipment Communication Controller
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
TCP	Transport Control Protocol. Stream oriented protocol used within TCP/IP
TCP/IP	Layered communication stack defined by IEEE802.3
TLS	Transport Layer Security
UDP	User Datagram Protocol. Frame oriented protocol used within TCP/IP
V2GTP	Vehicle to Grid Transfer Protocol

Table 50 Abbreviations

9 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com