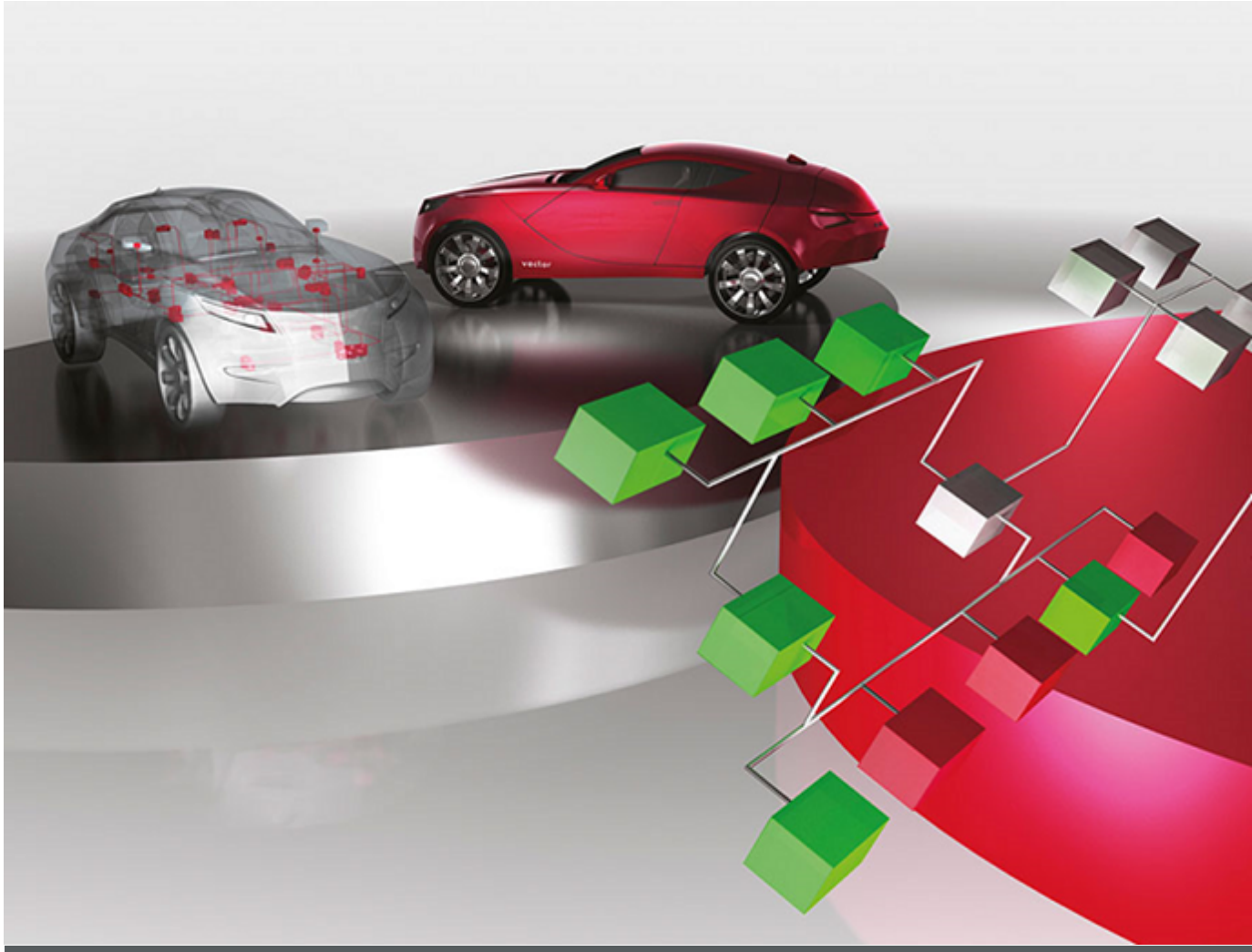
VECTOR



User Manual
CANoe .SmartCharging
DIN 70121 / ISO 15118

Version 11.0.3
English

Imprint

Vector Informatik GmbH
Holdäckerstraße 36
D-70499 Stuttgart

Table of Contents

1 Introduction

In this chapter you find the following information:

1.1 About this User Manual

To find information quickly

The user manual provides you the following access help:

> At the beginning of each chapter you will find a summary of the contents.

> In the header you can see in which chapter and paragraph you are located.

> In the footer you can see to which version the user manual applies.

> At the end of the user manual you will find an index, with whose help you will quickly find information.

Conventions

In the two following tables you will find the conventions used in the user manual regarding utilized spellings and symbols.

Style	Utilization
bold	Blocks, interface elements, and window and dialog names of the software. Accentuation of warnings and advice. **[OK]** Push buttons in brackets **File|Save** Notation for menus and menu entries
Windows	Legally protected proper names and side notes.
`Source code`	File name and source code.
Hyperlink	Hyperlinks and references.
<Ctrl>+<S>	Notation for shortcuts.

Symbol	Utilization
	Here you can obtain supplemental information.
	This symbol calls your attention to warnings.
	Here you can find additional information.
	Here is an example that has been prepared for you.
	Step-by-step instructions provide assistance at these points.
	Instructions on editing files are found at these points.
	This symbol warns you not to edit the specified file.

1.1.1 Certification

Certified Quality Management System	Vector Informatik GmbH has ISO 9001:2008 certification. The ISO standard is a globally recognized standard.

1.1.2 Warranty

Restriction of warranty	We reserve the right to modify the contents of the documentation or the software without notice. Vector disclaims all liabilities for the completeness or correctness of the contents and for damages which may result from the use of this documentation.

1.1.3 Support

You need support?	You can get through to our hotline at the phone number > Phone: +49 711 80670-200 > Email: support@de.vector.com > Online form: http://vector.com/support/

1.1.4 Trademarks

Protected trademarks	All brand names in this documentation are either registered or non registered trademarks of their respective owners. > See also chapter 9 Appendix: Copyright

2 Configuration of a Charge Point

In this chapter you find the following information:

2.1 Overview

Configuration file

The charge point parameters are configured using an XML file. The structure of this file is described by the supplied XML schema file “Schema_SCC_Config.xsd”, of which only the <EVSEConfiguration> element is relevant here. The meaning of each parameter is listed below; a more detailed description can be obtained from the schema file.

Reference: Please refer to the CANoe help for rules regarding the naming and placement of the XML configuration file.

2.2 General Parameters

V2GProtocolVersion	Transport Protocol version to be used (decimal), default = 1
V2GTimeout	Timeout for a connection to be considered inactive, in milliseconds.
SDPMessageDelay	Delay for sending the SECC Discovery Response in [ms], default = 10 Note: Use `SCC_SetMessageDelay()` to set the delay for V2G messages.
SessionStopOnError	If enabled (default), the charge point will accept session stop messages in active mode even if it is not in the corresponding state. This is a non-standard form of error handling.
StopOnVerificationError	If enabled, the charge point will stop the communication in active simulation mode when an invalid signature is received.
SchemaNamespace	Force the charge point to select a certain schema during the SupportedAppProtocol handshake. If not specified, the version with the highest priority is selected. This schema is also used by default for EXI decoding.
SchemaVersionMajor **SchemaVersionMinor**	Optionally limit SchemaNamespace to a specific version.
InvalidValueSigned	32 bit value to be returned when a missing optional message parameter is queried (default = -1, applies to types “long” and “float”)
InvalidValueUnsigned	32 bit value value >= 0 to be returned when a missing optional message parameter is queried (default = 0, applies to type “dword”)

2.3 Connection Parameters

EVSEListenPort	The port used for TCP. Set to “auto” for random port. (default: auto)
UseTLS	If set to 1, the SECC Discovery Response will be sent with security option “TLS” in active simulation mode and TLS will be enabled.

Note: IPv6 addresses must be configured in the usual notation. All variants of the IPv6 notation are supported.

Note: The actual IP address of the charge point node must be configured within CANoe (when the node stack is used) or via the real adapter (when the Windows stack is used) (see CANoe help). For multiple the real adapters, the address of the first adapter will be used. Using the parameter “EVSEIPAddress” overwrites this configuration.

2.4 V2G Parameters (active mode)

EVSEID	Unique ID of charge point (decimal)
MeterID	Meter ID within the charge point (string)
SAScheduleList	List of available tariffs and schedule (recent protocol versions). Note that DIN 70121 does not consider the element “SalesTariff”.
ServiceList	List of offered services and service parameters **As a deviation to the schema, each element** <Service> **may contain a** <ServiceParameterList> **for use with the Service Detail messages. See demo for an example.**
AllowContractPayment	If enabled, the payment option “Contract” is available, which corresponds to PnC mode (ISO 15118)
AllowExternalPayment	If enabled, the payment option “ExternalPayment” is available, which corresponds to EIM mode (ISO 15118)
AlwaysSendSASchedule	Forces the sending of the SASchedule element in every ChargeParameterDiscovery Response. If set to 0, it will only be sent when EVSEProcessing = TRUE, which corresponds to the ISO 15118 behavior. (default: 1)
EVSEMaxPower	Default maximum line power of charge point
EVSEMinVoltage	Default minimum supported line voltage
EVSEMaxVoltage	Default maximum supported line voltage
EVSEMinCurrent	Default minimum supported line current
EVSEMaxCurrent	Default maximum supported line current

EVSEMaxPhases	Default maximum supported number of phases (1 or 3)
EVSENominalVoltage `(ISO 15118)`	Default supported line voltage
EVSECurrentRegulation-Tolerance	Default absolute magnitude of the regulation tolerance
EVSEPeakCurrentRipple	Default peak-to-peak magnitude of the current ripple
CurrentDemandResDelay	Message delay >= 0 that applies only to the CurrentDemand response. If not specified, the value supplied with `SetMessageDelay()` ist used. This allows meeting the special time window requirements for this message.
CheckChargingProfile	If set to 1 (default), the charge point may reject ChargingProfiles according to [V2G-DC-267]. If set to 0, all profiles are accepted.

Note: The XML structure and data types of the parameters, i.e. the tariff and service lists, match their definition within the SCC specification in the context of the associated SCC messages. In particular, for float types, the float representation according to the specification must be used.

2.5 SLAC Parameters

UseSLAC	Configures default SLAC handling for both active and passive mode. 0 = SLAC messages are ignored 1 = SLAC message are answered
EVSEMACAddress	MAC address to be used as source address for SLAC frames. This should equal the address configured for the adapter / the CANoe node.
SLAC_MACFilter	Defines a MAC address whose messages will be ignored. This parameter can be contained multiple times in a configuration.
SLAC_AutoSetKey	Enables or disables all optimizations regarding the sending of CM_SetKey.Req, which includes early sending at measurement start or after disconnecting. If 0, the message is *only* sent at the end of each SLAC process.
SLAC_NID	Network ID (7 byte hexadecimal number)
SLAC_NMK	Network Membership Key (16 byte hexadecimal number). Setting this parameter defines a static NMK i.e. no new NMKs will be generated at runtime.

SLAC_ChipMACAddress	MAC address of the local QCA chip. If provided, it is used as source for attenuation messages, and to filter attenuation and link status messages, to ensure that no messages for other senders are interpreted.
SLAC_UseValidation	Enables or disable a positive reply to validation requests (note: the vehicle may choose to validate nonetheless) (default: 0)
SLAC_SendAmpMap-Confirmation	Enables the sending of CM_Amp_Map.Cnf messages each time a CM_Amp_Map.Req message is received. (default: 1)
SLAC_AttenuationRx	Attenuation of the Rx path to be subtracted from the AAG values (see [V2G-DC-569]). This value may be negative and have fractional digits, e.g. “-0.5”.
SLAC_LinkStatusPollingType	Determines the message(s) for querying the link status. Possible values are: “All” (0) : use both available messages “PlLnkStatus (1): use only VS_PL_Lnk_Status “NwInfo” (2): use only VS_Nw_Info
SLAC_LinkStatus-PollingInterval	Interval in ms for the sending of VS_PL_Lnk_Status or VS_Nw_Info messages to query the link status. If set to 0, polling is disabled (default: 100).
SLAC_LinkStatus-DebounceTime	Debounce time in ms for switching the link status (default: 0).
SLAC_UseSTPLinkStatus	Enables usage of STP_Link_Status (vendor specific) messages instead of VS_PL_Lnk_Status.
SLAC_ChipMagicAddress	Generic address used to address the QCA chip. (default: 00:B0:52:00:00:01)

QCA Chip Simulation

If no QCA7000/7005 (HomePlug Green PHY) chip is present, the charge point module has the possibility to generate the attenuation messages (CM_Atten_Profile.Ind) itself, which hold the attenuation values. Without these messages, the SLAC process cannot complete. The attenuation values are randomly created using a Gaussian distribution.

SLAC_ChipPresent	Enables or disables the sending of attenuation messages by default. The following parameters will be used as input data if activated.
SLAC_NumGroups	Number of attenuation groups
SLAC_AttenuationMean	Mean value of the probability distribution used for creating the attenuation values in dB
SLAC_AttenuationDeviation	Standard deviation of the probability distribution used for creating the attenuation values in dB

Note: The value of SLAC_ChipPresent is overridden by the CAPL function `SCC_SLAC_SetChipPresent()` (see section 3.7).

Note: The simulated chip provides attenuation values, but neither provides a link status nor answers to a VS_PL_Lnk_Status request. A change in the link status can be simulated with the function `SCC_SLAC_SetLinkStatus()` (see section 6.3)

2.6 Security Parameters

Usage of certificates

The following elements are only required for the ISO 15118 PnC profile. Certificates are referred by their Name property as displayed in the Vector Security Manager after importing the certificate.

Note: Only leaf certificates are referred in the XML config. Please make sure that the base certificates are also part of the Security Profile where full chains are required, as they will be resolved automatically using the Signer ID.

Reference: For the usage of the Vector Security Manager, please refer to the CANoe help.

ContractID / eMAID	Contract ID which is sent along with the certificates (only needed for Certificate Installation – else the ID sent by the vehicle will be used)
TLSHostCert	Certificate for hosting a TLS connection (the SECC's leaf certificate)
TLSUseClientAuthentication	Set to 1 to demand authentication from the EV during TLS handshake (default: 0)
MOSub2Cert	"Issuer certificate" (the first SubCertificate in the ContractSignatureCertChain) for signing the SalesTariff
SAProvisioningCert	Secondary actor certificate for signing CertificateInstallationRes and -UpdateRes
ContractCert	Contract certificate for sending to the EV during certificate update or installation, along with the corresponding chain
ContractCertPrivateKey	Private key to be sent to the EV during certificate update or installation

Note: Due to security-related restrictions, it is not possible to use the Contract Certificate's private key as configured in the Vector Security Manager to transfer it in a V2G message. It must be specified in the XML configuration file.

3 CAPL Interface to Charge Point

In This Chapter You Will Find the Following Information:

3.1 Callback Interface (SLAC messages)

Overview

Incoming SLAC messages are signaled by the charge point DLL through CAPL callbacks. All variable parameters are provided within the callback signature. Parameters which are defined constant by the DIN 70121:2014-12 specification are not included. If they are incorrect, the simulation DLL will not accept the message and will not call the respective function.

Constant fields

Further data that are not provided directly by the callback function can be queried by additional function calls, which are described in more detail in sections 3.5 and 6.2. Note that all variable data is included in the callback itself. The parameters queried by the additional functions are predefined by the DIN 70121:2014-12 specification.

CM_Slac_Parm_Req

Syntax	`void SCC_CM_Slac_Parm_Req ( byte RunId[], byte SourceMacAddress[] )`
Function	The callback is called as soon as a CM_Slac_Parm.Req message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetApplicationType** **SCC_SLAC_GetSecurityType**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender
Returns	-

CM_Start_Atten_Char_Ind

Syntax	`void SCC_CM_Start_Atten_Char_Ind ( byte RunId[], byte SourceMacAddress[], dword NumSounds, dword TimeOut, byte ForwardingSTA[] )`
Function	The callback is called as soon as a CM_Start_Atten_Char.Ind message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetApplicationType** **SCC_SLAC_GetSecurityType** **SCC_SLAC_GetRespType**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **NumSounds:** Number of M-Sounds transmitted during the SLAC process **TimeOut:** Timeout for transmission of M-Sounds in multiple of 100ms **ForwardingSTA:** MAC address where the measurement results shall be sent to
Returns	-

CM_MNBC_Sound_Ind

Syntax	`void SCC_CM_MNBC_Sound_Ind ( byte RunId[], byte SourceMacAddress[], dword Count )`
Function	The callback is called as soon as a CM_MNBC_Sound.Ind message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetApplicationType** **SCC_SLAC_GetSecurityType** **SCC_SLAC_GetSourceId** **SCC_SLAC_GetRandomValue**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **Count:** Countdown counter for number of Sounds remaining
Returns	-

Note: This indication can be used to set individual attenuation values for different vehicles. To implement this, call `SCC_SLAC_SetAttenuation()` (see 3.7) with values depending on the remote MAC address or the RunID of the process.

CM_Atten_Profile_Ind

Syntax	`void SCC_CM_Atten_Profile_Ind ( byte RunId[], byte SourceMacAddress[], byte PEVMAC[], dword NumGroups, byte AAG[] )`
Function	The callback is called as soon as a CM_Atten_Profile.Ind message is received.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **PEVMAC:** MAC Address of the vehicle **NumGroups:** Number of attenuation groups **AAG:** Average Attenuation Group (array length is indicated by the paremeter ‘NumGroups’)
Returns	-

CM_Atten_Char_Rsp

Syntax	`void SCC_CM_Atten_Char_Rsp ( byte RunId[], byte SourceMacAddress[], byte SourceAddress[] )`
Function	The callback is called as soon as a CM_Atten_Char.Rsp message is received. **SCC_SLAC_GetApplicationType** **SCC_SLAC_GetSecurityType** **SCC_SLAC_GetSourceId** **SCC_SLAC_GetResponseId** **SCC_SLAC_GetResult**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **SourceAddress:** MAC Address of the vehicle which initiates the SLAC process
Returns	-

CM_Validate_Req

Syntax	`void SCC_CM_Validate_Req ( byte RunId[], byte SourceMacAddress[], dword ListenTimer )`
Function	The callback is called as soon as a CM_Validate.Req message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetSignalType** **SCC_SLAC_GetResult**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **ListenTimer:** Time duration while the EVSE shall listen to BCB-toggles: 0x00 = 100 ms 0x01 = 200 ms
Returns	-

CM_SLAC_Match_Req

Syntax	`void SCC_CM_SLAC_Match_Req ( byte RunId[], byte SourceMacAddress[], byte PEVMacAddress[], byte EVSEMacAddress[] )`
Function	The callback is called as soon as a CM_SLAC_Match.Req message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetApplicationType** **SCC_SLAC_GetSecurityType** **SCC_SLAC_GetMVFLength** **SCC_SLAC_GetPEVAndEVSEId**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **PEVMacAddress:** MAC Address of the vehicle **EVSEMacAddress:** MAC Address of the EVSE
Returns	-

CM_Set_Key_Cnf

Syntax	`void SCC_CM_Set_Key_Cnf ( byte RunId[], byte SourceMacAddress[], dword Result )`
Function	The callback is called as soon as a CM_Set_Key.Cnf message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetCMSetKeyCnfData**.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **Result:** Result code: 0x00 = success 0x01 = failure 0x02-0xFF = reserved
Returns	-

VS_PL_Lnk_Status_Cnf

Syntax	`void SCC_VS_PL_Lnk_Status_Cnf ( byte SourceMacAddress[], dword MStatus, dword LinkStatus )`
Function	The callback is called as soon as a VS_PL_Lnk_Status.Cnf message is received. This is a response of the QCA9000 chip when using link status polling. (Additional data cannot be queried at the moment.)
Parameters	**SourceMacAddress:** MAC address of sender **MStatus:** MStatus code: 0x00 = success 0x01 = failure **LinkStatus:** Notification if the link is established: 0x00 = no link 0x01 = link
Returns	-

VS_Nw_Info_Cnf

Syntax	`void SCC_VS_Nw_Info_Cnf ( byte SourceMacAddress[], long NumAvLn, byte NID[] )`
Function	The callback is called as soon as a VS_Nw_Info.Cnf message is received. This is a response of the QCA9000 chip when using link status polling. (Additional data cannot be queried at the moment.)
Parameters	**SourceMacAddress:** MAC address of sender **NumAvLn:** Number of AVLANs (> 0 denotes an established link) **NID:** Network ID (7 byte)
Returns	-

3.2 Callback Interface (Vehicle Requests)

Overview

Incoming SCC Requests are signaled by the charge point DLL through CAPL callbacks. Further data that are not provided directly by the callback function can be queried by additional function calls, which are described in more detail in sections 3.5 and 6.2.

Caution: These functions can only be called within the associated callbacks.

SECCDiscoveryReq

Syntax	`void SCC_SECCDiscoveryReq ( dword Security, dword TransportProtocol )`
Function	The callback is called as soon as a SECC Discovery Request is received.
Parameters	**Security:** 1 for TLS, 0 for “no transport layer security” **TransportProtocol:** 0 for TCP, 0x10 for UDP
Returns	-

SupportedApp-ProtocolReq

Syntax	`void SCC_SupportedAppProtocolReq ( dword AppProtocolCount )`
Function	The callback is called as soon as a SupportedAppProtocol Request is received. The list entries must be queried via the separate help function **SCC_GetAppProtocolData**
Parameters	**AppProtocolCount:** Number of transmitted AppProtocol elements.
Returns	-

SessionSetupReq

Syntax	`void SCC_SessionSetupReq ( byte SessionID[], byte EVCCID[] )`
Function	The callback is called as soon as a Session Setup Request is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. If a new session is started, this ID is transferred. **EVCCID:** 6 byte MAC address of the vehicle
Returns	-

ServiceDiscoveryReq	**Syntax**	`void ServiceDiscoveryReq ( byte SessionId[], char ServiceScope[], char ServiceCategory[] )`
	Function	The callback is called as soon as a Service Discovery Request is received.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ServiceScope:** Typically designates a company/organization in the form of a URI. **ServiceCategory:** Type of requested services
	Returns	-

ServiceDetailReq	**Syntax**	`void SCC_ServiceDetailReq ( byte SessionID[], dword ServiceId )`
	Function	The callback is called as soon as a Service Detail Request is received.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ServiceId:** ID of the requested service (16 bit unsigned number).
	Returns	-

ServicePayment-SelectionReq	**Syntax**	`void SCC_ServicePaymentSelectionReq ( byte SessionID[], char SelectedPaymentOption[], long ServiceListCount )`
	Function	The callback is called as soon as a Service Payment Selection Request is received. The Request contains the selected services and the chosen method of payment. Further details that are transmitted in this request can be queried with **SCC_GetSelectedServiceID** **SCC_GetSelectedParameterSetID**
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **SelectedPaymentOption**: The selected payment option in form of a string. **ServiceListCount:** Number of selected services: Up to 8 entries are contained in the list. The list entries must be queried via separate help functions; see 3.5.
	Returns	-

PaymentDetailsReq	**Syntax**	`void SCC_PaymentDetailsReq ( byte SessionID[], char ContractID[] )`
	Function	The callback is called as soon as a Payment Details Request is received.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ContractID:** ID of the contract to which the certificate is assigned, max. 128 character string. (eMAID)
	Returns	-

AuthorizationReq

Syntax	`void SCC_AuthorizationReq ( byte SessionID[] )`
Function	The callback is called as soon as an Authorization / Contract Authentication Request is received. The challenge optionally transmitted in this request can be queried with **SCC_GetGenChallenge.**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF.
Returns	-

Certificate-InstallationReq

Syntax	`void SCC_CertificateInstallationReq ( byte SessionID[] )`
Function	The callback is called as soon as a Certificate Installation Request is received. The list of root certificate IDs can be queried with **SCC_GetOEMPRovisioningCertificate** **SCC_GetNumberOfRootCertificateIDs** **SCC_GetRootCertificateID** **SCC_GetDHPublicKey** `(DIN 70121)`
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF.
Returns	-

CertificateUpdateReq

Syntax	`void SCC_CertificateUpdateReq ( byte SessionID[], char ContractID[] )`
Function	The callback is called as soon as a Certificate Installation Request is received. The list of root certificate IDs can be queried with **SCC_GetNumberOfRootCertificateIDs** **SCC_GetRootCertificateID** **SCC_GetDHPublicKey** `(DIN 70121)`
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ContractID:** ID of the contract to which the certificate is assigned, max. 128 character string. (eMAID)
Returns	-

ChargeParameter-DiscoveryReq

Syntax (AC)	`void SCC_ChargeParameterDiscoveryReqAC ( byte SessionID[], float EAmount )`
Syntax (DC)	`void SCC_ChargeParameterDiscoveryReqDC ( byte SessionId[], double EVEnergyCapacity, double EVEnergyRequest)`
Function	The callback is called as soon as a Charge Parameter Discovery Request is received. Further details that are transmitted in this request can be queried with the following functions: **SCC_GetEnergyTransferType** **SCC_GetDepartureTime** **SCC_GetMaxVoltage** **SCC_GetMaxCurrent** **SCC_GetMinCurrent** (AC) **SCC_GetMaxPower** (DC) **SCC_GetFullSOC** (DC) **SCC_GetBulkOC** (DC) **SCC_GetMaxEntriesSAScheduleTuple** (ISO 15118)
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **EAmount (AC):** Energy required by the vehicle. **EVEnergyCapacity (DC):** Maximum supported power **EVEnergyRequest (DC):** Energy required by the vehicle.
Returns	-

PowerDeliveryReq

Syntax	`void SCC_PowerDeliveryReq ( byte SessionId[], long ChargeProfileCount, long ChargeState, long ScheduleID)`
Function	The callback is called as soon as a Power Delivery Request is received. With this request, the vehicle requests the charge point to switch on the current and to send the charging profile. Further details that are transmitted in this request can be queried with **SCC_GetChargingProfileData** **SCC_GetBulkChargingComplete** (DC) **SCC_GetChargingComplete** (DC)
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ChargingProfileCount:** Number of received charging profiles. **ChargeState:** 1 if start of charging is requested, 0 if stop of charging is requested, 2 if ReNegotiation is requested (only ISO 15118). Corresponds to “ReadyToChargeState” in DIN 70121. **ScheduleID:** ID of the chosen SAScheduleTuple
Returns	-

ChargingStatusReq

Syntax (AC)	`void SCC_ChargingStatusReq ( byte SessionID[] )`
Function	The callback is called as soon as a Charging Status Request is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF
Returns	-

MeteringReceiptReq

Syntax	`void SCC_MeteringReceiptReq ( byte SessionID[],`
	`  byte MessageSessionID[],`
	`  long ScheduleTableEntryID )`

Function	The callback is called as soon as a Metering Receipt Request is received. With this request, the vehicle confirms receipt of the metering information sent by the charge point. Further details that are transmitted in this request can be queried with the helper function **SCC_GetMeterInfoData**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF **MessageSessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF, as transferred within the message body. **ScheduleTableEntryID:** ID of the selected SAScheduleTuple
Returns	-

CableCheckReq

Syntax (DC)	`void SCC_CableCheckReq ( byte SessionID[] )`
Function	The callback is called as soon as a Cable Check Request is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF
Returns	-

PreChargeReq

Syntax (DC)	`void SCC_PreChargeReq ( byte SessionID[],` `  float TargetVoltage, float TargetCurrent )`
Function	The callback is called as soon as a PreCharge Request is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF **TargetVoltage:** Voltage demand **TargetCurrent:** Current demand
Returns	-

CurrentDemandReq

Syntax (DC)	`void SCC_CurrentDemandReq ( byte SessionID[], float TargetVoltage, float TargetCurrent, long BulkChargingComplete, long ChargingComplete )`
Function	The callback is called as soon as a Current Demand Request is received. Further details that are transmitted in this request can be queried with **SCC_GetMaxVoltage** **SCC_GetMaxCurrent** **SCC_GetMaxPower** **SCC_GetRemainingTimeToFullSoC** **SCC_GetRemainingTimeToBulkSoC**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF **TargetVoltage:** Voltage demand **TargetCurrent:** Current demand **BulkChargingComplete:** 1, if the main charging operation is complete; otherwise 0. **ChargingComplete:** 1, if the battery is completely charged; otherwise 0.
Returns	-

WeldingDetection-Req

Syntax (DC)	`void SCC_WeldingDetectionReq ( byte SessionID[] )`
Function	The callback is called as soon as a Welding Detection Request is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF
Returns	-

SessionStopReq

Syntax	`void SCC_SessionStopReq ( byte SessionID[], long Terminate )`
Function	The callback is called as soon as a Session Stop Request is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **Terminate:** 1 if charging status == terminate; for all other values: 0
Returns	-

3.3 Other Callback Functions

PreSendInd

Syntax	`void SCC_PreSendInd ( byte SessionID[], dword MessageID, char ResponseCode[] )`
Function	The callback is called in active mode before a response message is sent. It enables checking the message's response code and, if desired, overwriting it with another value. Additionally, overwriting the following parameters is supported: EVSEStatusCode EVSEIsolationStatus
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **MessageID:** Type of message to be sent (see section 9.1) **ResponseCode:** Response code string of the message
Returns	-

Caution: Only the parameters listed above can be overwritten during a PreSend indication. If any other parameter is changed, it will not be applied until the next message is sent.

SchemaSelectionInd

Syntax	`void SCC_SchemaSelectionInd ( char Namespace[], dword VersionMajor, dword VersionMinor )`
Function	Indicates that a schema has been chosen via the SupportedAppProtocol handshake. (The EVSE simulation automatically chooses a protocol from the vehicle's list based on the priority.)
Parameters	**Namespace:** Namespace string of the selected schema **VersionMajor:** Major version of the selected schema **VersionMinor:** Major version of the selected schema
Returns	-

3.4 Auxiliary Functions for Setting of Response Details

Context sensitivity

The following functions can be used to affect various message sent by the charge point. They require a session ID as a reference to the connection to be modified. However, for simplified usage the following applies:

> If only one connection is active, the parameter "SessionID"is not read. It may be set to an arbitrary value.

> If called from within a callback, the calling connection is always used as context. Again, the parameter "SessionID" is not read in this case.

SetFaultNotification

Syntax	`void SCC_SetFaultNotification ( char FaultCode[], char FaultMsg[] )`
Function	Sets the fault code and message for the next V2G message. The fault code is contained within the V2G header.
Parameters	**FaultCode:** Desired fault code, which must be a valid enum value according to the specification **FaultMsg:** Desired fault message string. If this string is empty, the optional message element is omitted.
Returns	0: Not successful 1: Successful

SetResponseCode

Syntax	`long SCC_SetResponseCode ( char ResponseCode[] )`
Function	Sets the return code of the next message to be sent.
Parameters	**ResponseCode:** Response code string, which must match a valid enum value
Returns	0: Not successful 1: Successful

SetProcessing

Syntax	`long SCC_SetProcessing ( long Processing )`
Function	Sets the processing status of the charge point. This status is used for controlling various message loops, where the vehicle will continue sending the same request until EVSEProcessing is set to “Finished” within the response message.
Parameters	**EVSEProcessing:** 0 if “Finished”, 1 if “Ongoing” resp. “Ongoing_WaitingForCustomerInteraction” `(ISO 15118)`
Returns	0: Not successful 1: Successful

Note: The charge point will never send EVSEProcessing = “Ongoing” unless this function is called. The special value “Ongoing_WaitingForCustomerInteraction” of ISO IS will automatically be applied according to [V2G2-854] if EVSEProcessing was set to 1.

SetMaxPower
SetMaxVoltage
SetMaxCurrent

Syntax	`long SCC_SetMaxPower ( float MaxPower )`
	`long SCC_SetMaxVoltage ( float MaxVoltage )`
	`long SCC_SetMaxCurrent ( float MaxCurrent )`

Function	Sets the limit for power / voltage / current. These limits are used in various messages for both AC and DC mode (the actual element name depends on the charging mode and the procotol version). Defaults can be set using the respective configuration file.
Parameters	**MaxPower / MaxVoltage / MaxCurrent:** Limit value to be set.
Returns	0: Not successful 1: Successful

SetMinCurrent

SetNominalVoltage

SetMinVoltage
SetCurrent-RegulationTolerance
SetPeakCurrent-Ripple

Syntax	`long SCC_SetMinCurrent ( float MinCurrent )`
Syntax (ISO 15118 AC)	`long SCC_SetNominalVoltage ( float NominalVoltage )`

Syntax (DC)	`long SCC_SetMinVoltage ( float MinVoltage )`
	`long SCC_SetCurrentRegulationTolerance ( float CurrentRegulationTolerance )`
	`long SCC_SetPeakCurrentRipple ( float PeakCurrentRipple)`

Function	Sets lower limits and other electrical values for the message ChargeParameterDiscoveryRes. These limits are used in various messages for both AC and DC mode (the actual element name depends on the charging mode and the procotol version). Defaults can be set using the respective configuration file.
Parameters	**Parameter:** Value to be set.
Returns	0: Not successful 1: Successful

SetPresentVoltage
SetPresentCurrent

Syntax	`long SCC_SetPresentVoltage ( float PresentVoltage )` `long SCC_SetPresentCurrent ( float PresentCurrent )`
Function	Sets the current voltage / current output for the respective connection. These values are used in various DC messages, and for the MeterInfo in AC mode. If no values are set, the charge point will automatically calculate defaults.
Parameters	**PresentVoltage / PresentCurrent:** Value to be set.
Returns	0: Not successful 1: Successful

SetEnergy-ToBeDelivered

Syntax (DC)	`long SCC_SetEnergyToBeDelivered ( float EnergyToBeDelivered )`
Function	Sets the target energy value, which is sent in the ChargeParameterDiscoveryRes message.
Parameters	**EnergyToBeDelivered:** Value to be set.
Returns	0: Not successful 1: Successful

SetGenChallenge

Syntax	`long SCC_SetGenChallenge ( byte GenChallenge[] )`
Function	Sets the challenge for the PaymentDetailsRes message, which is randomly generated otherwise.
Parameters	**GenChallenge:** Value to be set. Array length must be 16 byte.
Returns	0: Not successful 1: Successful

SetSelectedSchema

Syntax	`long SCC_SetSelectedSchema ( long SchemaID )`
Function	Sets the ID of the schema to be selected in the SupportedAppProtocolRes message. This overrides the automatic selection done based on the vehicle's priority values.
Parameters	**SchemaID:** ID of the selected schema, based on the vehicle's list
Returns	0: Not successful 1: Successful

Note: Due to technical reasons, the schema ID must belong to a schema actually offered by the vehicle (the charge point cannot continue with an undefined schema).

Charge Loop

The following functions can be used during the actual charging operation:

SetMeterReading

Syntax (AC)	`long SCC_SetMeterReading( float CurrentMeter )`
Function	Sets the current consumption in the charge point.
Parameters	**CurrentMeter:** Meter reading to be set in Wh
Returns	0: Not successful 1: Successful

SetReceiptRequired

Syntax	`long SCC_SetReceiptRequired ( long ReceiptRequired )`
Function	Sets the flag "ReceiptRequired", and adapts the simulation state to await a MeteringReceiptReq next. Although MeteringReceipt is a specific message for PnC mode, this is also possible when using EIM mode.
Parameters	**ReceiptRequired:** 1 if a MeteringReceiptReq is expected next, 0 if another ChargingStatusReq is expected
Returns	0: Not successful 1: Successful

3.5 Auxiliary Functions for Querying of Request Details

Note: The functions can be called only within the assigned callback. They are used to query the details of a request.

SLAC_GetRandom-Value

Syntax	`void SCC_SLAC_GetRandomValue ( byte Rnd[] )`
Returns	The random value transmitted with CM_MNBC_Sound.Ind, to the output buffer (16 byte)

SLAC_GetResult

Syntax	`long SCC_SLAC_GetResult ( )`
Function	Queries the result code (from different SLAC messages)
Parameters	-
Returns	In case of CM_Atten_Char.Rsp: 0x00 = Success 0x01-0xFF = Reserved In case of CM_Validate.Req: 0x00 = Not Ready 0x01 = Ready 0x02 = Success 0x03 = Failure 0x04 = Not Required 0x05-0xFF = Reserved

GetMsgHeaderFault Notification

Syntax	`long SCC_GetMsgHeaderFaultNotification ( char FaultCode[], char FaultMsg[] )`
Returns	0 if no fault code is contained in the V2G header 1 if a fault code is contained in the V2g header In the latter case, fault code and optionally fault message are copied to the output buffers. Missing fault message results in an empty string.

GetMessageBody-IdAttr

Syntax	`long SCC_GetMessageBodyIdAttr ( char Id[] )`
Returns	The Id attribute of the message body, if existing (to the output buffer)

GetDC_EVStatus

Syntax (DIN 70121)	`void SCC_GetDC_EVStatus ( long& EVReady, long& EVCabinConditioning, long& EVRESSConditioning, char EVErrorCode[], long& EVRESSSOC)`
Syntax (ISO 15118)	`void SCC_GetDC_EVStatus ( long& EVReady, char EVErrorCode[], long& EVRESSSOC)`
Returns	Returns the DC status bits of the vehicle and its error code, if contained in the message.

SupportedApp-ProtocolReq	**Syntax**	`void SCC_GetAppProtocolData ( long Index, char ProtocolNamespace[], long& VersionMajor, long& VersionMinor, long& SchemaID, long& Priority )`
	Returns	Protocol namespace string (100 characters) and all other parameters contained in the AppProtocol element with the specified list index.

ServicePayment-SelectionReq	**Syntax**	`long SCC_ GetSelectedServiceID ( long Index )`
	Returns	ID of the selected service with the specified list index < ServiceCount.
	Syntax	`long SCC_ GetSelectedParameterSetID ( long Index )`
	Returns	ID of the selected service's parameter set with the specified list index < ServiceCount.

Contract-AuthenticationReq	**Syntax**	`void SCC_GetGenChallenge ( byte GenChallenge[] )`
	Returns	The challenge generated by the charge point (to a 16 byte output buffer).

Certificate-InstallationReq	**Syntax**	`void SCC_GetOEMPRovisioningCertificate ( char Certificate[] )`
	Returns	Reads the OEM provisioning certificate from the target certificate as base64 string (to the output buffer).

CertificateUpdateReq Certificate-InstallationReq	**Syntax**	`long SCC_GetNumberOfRootCertificateIDs ( )`
	Returns	The number of root certificates IDs transmitted in the message's ListOfRootCertificateIDs
InstallationReq	**Syntax**	`void SCC_GetRootCertificateID ( long Index, char IdOrIssuer [], byte SerialNumber[], long& SerialNumberLength )`
	Returns	The content of the RootCertificateID element with the specified index < GetNumberOfRootCertificateIDs(). For `ISO 15118`: The X509IssuerName to the buffer **IdOrIssuer** The X509SerialNumber to the buffer **SerialNumber** The byte length of X509SerialNumber via the referenceo **SerialNumberLength** Else: TheRootCertificateId string to the buffer **IdOrIssuer**, while the other parameters are unused

ChargeParameter-DiscoveryReq	**Syntax**	`long SCC_GetEnergyTransferType ( char[] EnergyTransferType )`
	Returns	Type of requested power supply: AC_Charging = 0, DC_Charging = 1 The EnergyTransferType resp -Mode string is additionally written to the output buffer. If this is not required, an empty string can be transferred.
	Syntax	`float SCC_GetMaxVoltage ( )`
	Returns	Maximum line voltage of vehicle (AC: “EVMaxVoltage”, DC: “EVMaximumVoltageLimit”).
	Syntax	`float SCC_GetMaxCurrent ( )`
	Returns	Maximum line current of vehicle (AC: “EVMaxCurrent”, DC: “EVMaximumCurrentLimit”).
	Syntax (AC)	`float SCC_GetMinCurrent ( )`
	Returns	Minimum line current of vehicle.
	Syntax (DC)	`float SCC_GetMaxPower ( )`
	Returns	Maximum power of vehicle (AC: “EVMaxPower”, DC: “EVMaximumPowerLimit”).
	Syntax (DC)	`long SCC_GetFullSOC ( )`
	Returns	Charging status in which the vehicle regards the battery as fully charged, in percent.
	Syntax (DC)	`long SCC_GetBulkSOC( )`
	Returns	Charging status in which the vehicle regards a bulk charging process as finished, in percent.
	Syntax	`dword SCC_GetDepartureTime( )`
	Returns	End of charging time as offset in seconds from the time this message is sent.
	Syntax (ISO 15118)	`long SCC_GetMaxEntriesSAScheduleTuple( )`
	Returns	Maximal allowed number of entries in the SAScheduleTuple

PowerDeliveryReq	**Syntax**	`long SCC_GetChargingProfileData ( long Index, dword& Start, float& MaxPower, long& MaxNumberOfPhases )`
	Returns	Content of the ChargingProfile with the specified index via references. **Start:** Start time of charging, in seconds from the time the request is sent. **MaxPower:** Maximum line power of charging profile **MaxNumberOfPhases:** Maximum number of phases of charging profile

	Syntax	`long SCC_GetBulkChargingComplete ( )`
	Returns	Get the flags BulkChargingComplete from DC_EVPowerDeliveryParameter, if contained
	Syntax	`long SCC_GetChargingComplete ( )`
	Returns	Get the flags ChargingComplete from DC_EVPowerDeliveryParameter

MeteringReceiptReq	**Syntax**	`void SCC_GetMeterInfoData ( char MeterID, float& MeterReading, byte SigMeterReading, long& MeterStatus, long& TMeter )`
	Returns	Data assigned to the meter: ID (32 characters), current meter reading in [Wh], signature, status and timestamp in UNIX format Length of signature is dependent on the encryption algorithm.

CurrentDemandReq	**Syntax**	`float SCC_GetMaxPower ( )`
	Returns	Maximum power of vehicle (“EVMaximumPowerLimit”).
	Syntax	`float SCC_GetMaxVoltage ( )`
	Returns	Maximum line voltage of vehicle (“EVMaximumVoltageLimit”).
	Syntax	`float SCC_GetMaxCurrent ( )`
	Returns	Maximum line current of vehicle (“EVMaximumCurrentLimit”).
	Syntax	`float SCC_GetRemainingTimeToFullSoC ( )`
	Returns	Remaining time until full charging condition in seconds.
	Syntax	`float SCC_GetRemainingTimeToBulkSoC ( )`
	Returns	Remaining time to end of a bulk charging operation in seconds.

3.6 Status Queries

GetDCStatusCode	**Syntax (DC)**	`void SCC_GetDCStatusCode ( byte SessionID[], char StatusCode[] )`
	Function	Outputs the DC status in string form.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF **StatusCode:** String buffer to which the status code is written.
	Returns	-

GetConnectionCount	**Syntax**	`long SCC_GetConnectionCount ( )`
	Function	Outputs the number of active SCC connections.
	Parameters	-
	Returns	Number of connections.

GetSessionId

Syntax	`void SCC_GetSessionId ( byte SessionID [] )`
Function	Outputs the SessionID of a connection.
Parameters	**SessionID:** 8-byte long array to which the SessionID is written.
Returns	-

SLAC_GetKeyData

Syntax	`void SCC_SLAC_GetKeyData ( byte NID[], byte NMK[] )`
Function	Gets the currently stored NID and NMK values (when automatic SLAC is used).
Parameters	**NID:** Network ID (7 byte hexadecimal number) **NMK:** Network Membership Key (16 byte)
Returns	-

3.7 Controlling the Charge Point Behavior

EVSEStatus

The following functions change the statuses of the charge point.

SetRCD

Syntax (AC)	`long SCC_SetRCD ( long RCD )`
Function	The function sets the residual current device.
Parameters	**RCD:** Open, error = 1 Closed, no error = 0
Returns	0: Not successful 1: Successful

SetIsolationStatus

Syntax (DC)	`long SCC_SetIsolationStatus ( char IsolationStatus[] )`
Function	Sets the isolation status enum.
Parameters	**IsolationStatus:** String buffer that is written to the isolation status. The string should be a valid enum value according the schema.
Returns	0: Not successful 1: Successful

SetEVSENotification

Syntax	`long SCC_SetEVSENotification ( char EVSENotification[] )`
Function	Sets the EVSE notification enum.
Parameters	**EVSENotification:** String buffer that is written to the EVSE notification. The string should be a valid enum value according to the schema.
Returns	0: Not successful 1: Successful

SetNotification-MaxDelay

Syntax	`long SCC_SetNotificationMaxDelay ( long MaxDelay )`
Function	Sets the maximally allowed delay time for the vehicle to react on the provided notification.
Parameters	**MaxDelay:** Target delay time.
Returns	0: Not successful 1: Successful

SetDCStatusCode

Syntax (DC)	`long SCC_SetDCStatusCode ( char StatusCode[] )`
Function	Sets the DC status.
Parameters	**StatusCode:** String buffer that is written to the status code.
Returns	0: Not successful 1: Successful

ResetDCStatusCode

Syntax (DC)	`long SCC_ResetDCStatusCode ( )`
Function	Sets the DC status to its (protocol version dependent) default value.
Parameters	-
Returns	0: Not successful 1: Successful

PaymentOptions

Syntax	`long SCC_SetContractPaymentAllowed ( dword Allowed )` `long SCC_SetExternalPaymentAllowed ( dword Allowed )`
Function	Changes the available PaymentOptions to be sent in the ServiceDiscovery response message.
Parameters	**Allowed:** 1 to allow the PaymentOption, 0 to disallow. The other option will automatically be activated if an option is disallowed.
Returns	0: Not successful 1: Successful

SetShutdown-Request

Syntax	`long SCC_SetShutdownRequest ( long ShutdownRequest )`
Function	Demands the stop of the charging session from the vehicle, using the appropriate mechanism for the active schema version, or withdraws this request.
Parameters	**ShutdownRequest:** 1 to demand a shutdown, 0 to withdraw a shutdown request
Returns	0: Not successful 1: Successful

Note: You can still initiate a shutdown using the specific parameter of the target schema version instead of using this convenience function.

StopSession

Syntax	`long SCC_StopSession ( )` `long SCC_StopSession ( dword CloseTcpConnection )`
Function	The function stops a connection immediately. If it is used within a message callback, a response to the message is not sent.
Parameters	**CloseTcpConnection:** If 1, an additional TCP close is executed
Returns	0: Not successful 1: Successful

SuspendTx

Syntax	`void SCC_SuspendTx ( long NumberOfMessages )`
Function	Skips the sending of the following messages depending on the parameter value.
Parameters	**NumberOfMessages:**

Value	Behavior
-1	Sending of all further messages is suspended.
0	Resume. Starting from current state messages are send.
>0	The following “NumberOfMessages” are suspended.

Returns	-

SLAC

The following functions apply to the SLAC process.

SLAC_SetChip-Present

Syntax	`long SCC_SLAC_SetChipPresent` `( dword ChipPresent )`
Function	Configures the SLAC protocol to run with a real QCA7000 chip, or without it. This toggles the sending of artificial attenuation data.
Parameters	**ChipPresent:** 1 to disable chip simulation, 0 to enable-
Returns	0: Not successful 1: Successful

Note: This function overrides the configuration parameter <SLAC_ChipPresent> (see section 2.5). The supplied EV / EVSE configurations for VT8770 are already configured to use a real chip - there is no need to adapt these.

SLAC_Generate-ApplyKey

Syntax	`long SCC_SLAC_GenerateApplyKey ( )`
Function	Immediately sends a CM_Set_Key.Req message with a new NMK and NID. If a NMK is defined in the XML configuration file, this NMK is permanently discarded.
Parameters	-
Returns	0: Not successful 1: Successful

SLAC_Set-Attenuation

Syntax	`long SCC_SLAC_SetAttenuation ( float AttenuationMean, float AttenuationDev )`
Function	Changes the probability distribution used to create attenuation characteristics when simulating a QCA chip. The values apply to all SLAC sessions, yet may be set each time an M-Sound is received using the indication SCC_CM_MNBC_Sound_Ind.
Parameters	**AttenuationMean:** Attenuation Mean Value in dB **AttenuationDev:** Standard deviation of the distribution in dB
Returns	0: Not successful 1: Successful

SLAC_Set-AttenuationRx

Syntax	`long SCC_SLAC_SetAttenuationRx ( float AttenuationRx )`
Function	Sets the attenuation of the Rx path, which is subtracted from the AAG values. This is can also be used to influence the calculation of attenuation values when a real chip is used, i.e. the attenuation may be artificially raised or lowered.
Parameters	**AttenuationRx:** Attenuation value
Returns	0: Not successful 1: Successful

SCC_SLAC_Set-ToggleNum

Syntax	`long SCC_SLAC_SetToggleNum ( int ToggleNum )`
Function	Sets the number of toggles for the next CM_Validate.Cnf message. If this function is not used, either the value from the XML configuration or the default (2) is used.
Parameters	**ToggleNum:** Number of received toggles (max. 255). Real systems use toggle numbers <= 3.
Returns	0: Not successful 1: Successful

Note: The number of toggles is only applied to the second CM_Validate.Cnf in a validation sequence. The first one is specified to always have ToggleNum = 0.

4 Configuration of a Vehicle

In this chapter you find the following information:

4.1 Overview

Configuration file

The vehicle parameters are configured using an XML file. The structure of this file is described by the supplied XML schema file “Schema_SCC_Config.xsd”, of which only the <PEVConfiguration> element is relevant here. The meaning of each parameter is listed below; a more detailed description can be obtained from the schema file.

Reference: Please refer to the CANoe help for rules regarding the naming and placement of the XML configuration file.

4.2 General Parameters

V2GProtocolVersion	Transport Protocol version to be used (decimal), default = 1
V2GTimeout	Timeout for a connection to be considered inactive, in milliseconds.
ErrorTimeout	Timeout after protocol errors in milliseconds before the vehicle restarts the protocol.
V2GTimeoutSettings	Behavior in case of Timeouts according to norm specification (EVCCMsgTimeout, EVCCCableCheckTimeout, EVCCPreChargeTimeout). In case of timeout: “NoSpecTimeOut”: Timeouts are igored “NoBreakOnSpecTimeout”: callback and Write window warning “BreakOnSpecTimeout”: additionally closes the TCP connection
V2GRetries	Number of retries for the communication after connection has been closed.
ChargePointDiscoveryRetries	Number of retries when sending a SECC Discovery Request (default: 4, which equals to 5 request messages in total)
ChargePointDiscoveryTimeout	Time between SDP retries, in milliseconds (default: 1000)
SessionStopOnError	If enabled, the vehicle will send a SessionStopReq message before terminating the protocol due to errors. This is a non-standard form of error handling. (default: 0)
StopOnVerificationError	If enabled, the charge point will stop the communication in active simulation mode when an invalid signature is received.
ServiceScope	Service Scope string to use for the Service Discovery Request (ISO 15118)
PreferredPaymentOption	Payment option to select if offered by the charge point (“Contract” or “ExternalPayment”). “Contract” selects PnC mode.

ScheduleEntrySelectionMode	Defines which schedule entry (SAScheduleTuple) is selected (“first” for the first tuple, “latest” to select the tuple with the highest ID)
SchemaNamespace **SchemaVersionMajor** **SchemaVersionMinor**	Default schema to be requested in the SupportedAppProtocolReq message, which determines the target protocol version. Also defines the schema to use if no protocol handshake takes place (free sending).
PublishAllSchemas	If enabled, the vehicle will send all other supported schemas in the SupportedAppProocolReq, in addition to the chosen schema.
InvalidValueSigned	32 bit value to be returned when a missing optional message parameter is queried (default = -1, applies to types “long” and “float”)
InvalidValueUnsigned	32 bit value value >= 0 to be returned when a missing optional message parameter is queried (default = 0, applies to type “dword”)

4.3 Connection Parameters

PEVUDPPort	The port used for UDP. Set to “auto” for random port. (default: auto)
PEVTCPPort	The port used for TCP. Set to “auto” for random port. (default: auto)
UseTLS	If set to 1, the SECC Discovery Request will be sent with security option “TLS” and the vehicle will try to establish a TLS connection.

4.4 V2G Parameters (active mode)

DefaultChargingMode	Charging mode (energy transfer type / mode) requested by default
EnergyCapacity	Battery capacity in Wh
PublishDCEnergyRequest	Enables or disables sending of energy capacity and energy request values in the ChargeParameterDiscovery Request (DC)
FullSOC	Percentage (integer) of battery level that indicates full charge (default = 99)
BulkSOC	Percentage (integer) of battery level that indicates the end of a bulk charge session (default = 80)
PEVMaxPower	Default maximum line power of vehicle
PEVMinVoltage	Default minimum supported line voltage

PEVMaxVoltage	Default maximum supported line voltage
PEVMinCurrent	Default minimum supported line current
PEVMaxCurrent	Default maximum supported line current
PEVMaxPhases	Default maximum supported number of phases (1 or 3)
ChargeLoopInterval	Time for one charging cycle (interval between consecutive request messages)
WeldingDetectionCount	Number of WeldingDetectionReq messages to be sent (default: 1)
PreChargeCount	Number of PreChargeReq messages to be sent. If set to “AUTO” (default), the vehicle will continue sending PreCharge messages until voltage and current provided by the EVSE match the requested values.
PreChargeTolerance	Tolerance in [%] for matching the EVSE’s voltage and the target voltage during the PreCharge loop (default = 5%)
ChargingProfile	Charging profile information for the Power Delivery Request message. Note that the SAScheduleTupleID can be overwritten via CAPL with the function `SetSelectedScheduleTableEntry` (see section 5.4).
MaxEntriesSAScheduleTuple	Maximum number of SAScheduleTuples (min. 12; remove to disable sending)
DepartureTime	Planned time of departure in seconds from now (remove to disable sending)

Note: The XML structure and data types of the configuration parameters match their definition within the SCC specification in the context of the associated SCC messages. In particular, for float types, the float representation according to the specification must be used.

4.5 SLAC Parameters

UseSLAC	Configures default SLAC handling for both active and passive mode. 0 = SLAC disabled 1 = SLAC enabled 2 / auto = use SLAC depending on link status (not recommended due to QCA chip instabilities)
PEVMACAddress	MAC address to be used as source address for SLAC frames. This should equal the address configured for the adapter / the CANoe node.
SLAC_MACFilter	Defines a MAC address whose messages will be ignored. This parameter can be contained multiple times in a configuration.

SLAC_AttnThresholdMin	Maximum averaged attenuation to consider a Charge Point as “Found”
SLAC_AttnThresholdMax	Maximum averaged attenuation (>SLAC_AttnThresholdMin) to consider a Charge Point as “Potentially found”. Only relevant for the validation process.
SLAC_RunID	User configured ID (8 byte) for SLAC sessions. If not defined, IDs will be random.
SLAC_UseValidation	Enables or disable the possibility to validate Charge Points considered “potentially found”. (default: 1)
SLAC_ForceValidation	Validate even if EVSE answers with “not required” (only in combination with SLAC_UseValidation) (default: 0)
SLAC_ValidationTimer	Timer value for the validation, if enabled, in multiple of 100 milliseconds.
SLAC_SendAmpMap-Confirmation	Enables the sending of CM_Amp_Map.Cnf messages each time a CM_Amp_Map.Req message is received. (default: 1)
SLAC_LinkStatusPollingType	Determines the message(s) for querying the link status. Possible values are: “All” (0) : use both available messages “PlLnkStatus (1): use only VS_PL_Lnk_Status “NwInfo” (2): use only VS_Nw_Info
SLAC_LinkStatus-PollingInterval	Interval in ms for the sending of VS_PL_Lnk_Status messages to query the link status. If set to 0, polling is disabled. (default: 100)
SLAC_LinkStatus-DebounceTime	Debounce time in ms for switching the link status (default: 0).
SLAC_UseSTPLinkStatus	Enables usage of STP_Link_Status (vendor specific) messages instead of VS_PL_Lnk_Status.
SLAC_DelayTime	Duration in milliseconds that the vehicle waits after the SLAC process, before sending the SECC Discovery Request (default: 0). Only applied if link status polling is disabled.
SLAC_ChipMACAddress	MAC address of the local QCA chip. If provided, it is used to filter link status messages, to ensure that no messages for other senders are interpreted.
SLAC_ChipMagicAddress	Generic address used to address the QCA chip. (default: 00:B0:52:00:00:01)

Note: A QCA7000/7005 chip may need to reset after setting a new NMK. During this interval, no communication is possible. You can either use link status polling to automatically continue the communication when the link is established, or simply bridge this gap by using SLAC_DelayTime.

Note: Although the validation messages can by sent by the EV simulation, no actual validation takes place at the moment. The simulation will always accept the result sent by the Charge Point.

Example: XML example files accompany the supplied Demo configuration. You can use them as the basis for your own configuration.

4.6 Security Parameters

Usage of certificates

The following elements are only required for the ISO 15118 PnC profile. Certificates are referred by their Name property as displayed in the Vector Security Manager after importing the certificate.

Note: Only leaf certificates are referred in the XML config. Please make sure that the base certificates are also part of the Security Profile where full chains are required, as they will be resolved automatically using the Signer ID.

Reference: For the usage of the Vector Security Manager, please refer to the CANoe help.

ContractID / eMAID	Contract ID which is sent along with the certificates (may be overwritten by a CertificateInstallation response).
TLSClientCert	Certificate for initiating a TLS connection (only if Client Authentication is required)
TLSHostCert	Certificate of the TLS host, if known
ContractCert	Contract certificate of the vehicle
MOSub2Cert	“Issuer certificate” (the first SubCertificate in the ContractSignatureCertChain) for verification of the SalesTariff
OEMProvisioningCert	OEM provisioning certificate to be used for the certificate installation process
ListOfRootCertificateIDs	List of root certificate IDs to be sent during certificate installation

5 CAPL Interface to Vehicle

In this chapter you will find the following information:

5.1 Callback Interface (SLAC messages)

Overview

Incoming SLAC messages are signaled by the vehicle DLL through CAPL callbacks. All variable parameters are provided within the callback signature. Parameters which are defined constant by the DIN 70121:2014-12 specification are not included. If they are incorrect, the simulation DLL will not accept the message and will not call the respective function.

Constant fields

Further data that are not provided directly by the callback function can be queried by additional function calls, which are described in more detail in sections 5.5 and 6.2. Note that all variable data is included in the callback itself. The parameters queried by the additional functions are predefined by the DIN 70121:2014-12 specification.

CM_SLAC_Parm_Cnf

Syntax	`void SCC_CM_SLAC_Parm_Cnf ( byte RunId[], byte SourceMacAddress[], dword NumSounds, dword TimeOut, byte ForwardingSTA[] )`
Function	The callback is called as soon as a CM_SLAC_Parm.Cnf message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetMSoundTarget** **SCC_SLAC_GetRespType**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **NumSounds:** Number of M-Sounds to be transmitted **TimeOut:** Timeout for transmission of M-Sounds in multiple of 100ms **ForwardingSTA:** MAC address where the measurement results shall be sent to
Returns	-

CM_Atten_Char_Ind

Syntax	`void SCC_CM_Atten_Char_Ind ( byte RunId[], byte SourceMacAddress[], byte SourceAddress[], dword NumSounds, dword NumGroups, byte AAG[] )`
Function	The callback is called as soon as a CM_Atten_Char.Ind message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetApplicationType** **SCC_SLAC_GetSecurityType** **SCC_SLAC_GetSourceId** **SCC_SLAC_GetResponseId**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **SourceAddress:** MAC Address of the EV which initiates the SLAC process **NumSounds:** Number of M-Sounds used to generate the ATTEN_PROFILE **NumGroups:** Number of attenuation groups **AAG:** Average Attenuation Group (array length is indicated by the paremeter 'NumGroups')
Returns	-

CM_Validate_Cnf

Syntax	`void SCC_CM_Validate_Cnf ( byte RunId[], byte SourceMacAddress[], dword Result, dword ToggleNum )`
Function	The callback is called as soon as a CM_Validate.Cnf message is received. Further details (signal type) that are transmitted in this request can be queried with **SCC_SLAC_GetSignalType.**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **Result:** Result code: 0x00 = Not Ready 0x01 = Ready 0x02 = Success 0x03 = Failure 0x04 = Not Required 0x05-0xFF = Reserved **ToggleNum:** Number of BC-edges detected by the EVSE
Returns	-

CM_SLAC_Match_Cnf

Syntax	`void SCC_CM_SLAC_Match_Cnf ( byte RunId[], byte SourceMacAddress[], byte PEVMAC[], byte EVSEMAC[], byte NID[], byte NMK[] )`
Function	The callback is called as soon as a CM_SLAC_Match.Cnf message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetApplicationType** **SCC_SLAC_GetSecurityType** **SCC_SLAC_GetMVFLength** **SCC_SLAC_GetPEVAndEVSEId**
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **PEVMAC:** PEV MAC Address **EVSEMAC:** EVSE MAC Address **NID:** Network ID given by the CCo (EVSE) **NMK:** Private NMK of the EVSE (random value)
Returns	-

CM_Set_Key_Cnf

Syntax	`void SCC_CM_Set_Key_Cnf ( byte RunId[], byte SourceMacAddress[], dword Result )`
Function	The callback is called as soon as a CM_Set_Key.Cnf message is received. Further details that are transmitted in this request can be queried with **SCC_SLAC_GetCMSetKeyCnfData**.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **Result:** Result code: 0x00 = success 0x01 = failure 0x02-0xFF = reserved
Returns	-

VS_PL_Lnk_Status_Cnf

Syntax	`void SCC_VS_PL_Lnk_Status_Cnf ( byte SourceMacAddress[], dword MStatus, dword LinkStatus )`
Function	The callback is called as soon as a VS_PL_Lnk_Status.Cnf message is received. This is the response of the QCA9000 chip when using link status polling. (Additional data cannot be queried at the moment.)
Parameters	**SourceMacAddress:** MAC address of sender **MStatus:** MStatus code: 0x00 = success 0x01 = failure **LinkStatus:** Notification if the link is established: 0x00 = no link 0x01 = link
Returns	-

5.2 Callback Interface (Charge Point Responses)

Overview

Incoming SCC Responses are signaled by the vehicle DLL through CAPL callbacks. Further data that are not provided directly by the callback function can be queried by additional function calls, which are described in more detail in section 5.5.

Caution: These functions can only be called within the associated callbacks.

Response codes

All response callbacks contain a Boolean parameter in their signatures that specifies the type of response code. With the function

SCC_GetResponseCodeString

the specific ResponseCode can be queried as a character string.

Note: This applies to all response callbacks with “ResponseCode” parameter and is therefore no longer stated for these callbacks.

SECCDiscoveryRes

Syntax	`void SCC_SECCDiscoveryRes ( dword Security, dword TransportProtocolType )`
Function	The callback is called as soon as an SECC Discovery Response is received. Further details that are transmitted in this request can be queried with **SCC_GetEVSEIP** **SCC_GetEVSEPort**
Parameters	**Security:** 1 for TLS, 0 for “no transport layer security” **TransportProtocol:** 0 for TCP, 0x10 for UDP
Returns	-

SupportedApp-ProtocolRes

Syntax	`void SCC_SupportedAppProtocolRes ( long ResponseCode, dword SchemaID )`
Function	The callback is called as soon as a Supported App Protocol Response is received.
Parameters	**ResponseCode:** 1 if “OK”, 0 if “FAILED” **SchemaID:** The ID of the schema that was selected by the charge point.
Returns	-

SessionSetupRes

Syntax	`void SCC_SessionSetupRes ( byte SessionID[], long ResponseCode, char EVSEID[] )`
Function	The callback is called as soon as a Session Setup Response is received. Further details that are transmitted in this request can be queried with **SCC_GetTimestamp**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **EVSEID:** ID of charge point
Returns	-

ServiceDiscoveryRes

Syntax	`void SCC_ServiceDiscoveryRes ( byte SessionID[], long ResponseCode, long ServiceCount )`
Function	The callback is called as soon as a Service Discovery Response is received. Further details that are transmitted in this request can be queried with the following functions: **SCC_GetServiceData** **SCC_GetPaymentOptions** **SCC_GetEnergyTransferType** `(DIN 70121)` **SCC_GetEnergyTransferModeCount** `(ISO 15118)` **SCC_GetEnergyTransferMode** `(ISO 15118)`
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **ServiceCount:** Number of ServiceTags or Services transmitted
Returns	-

Note: The element “ChargeService”, which is not part of the service list, is contained in the ServiceCount and referenced using index 0.

ServiceDetailRes

Syntax	`void SCC_ServiceDetailRes ( byte SessionID[], long ResponseCode, dword ServiceID, long ParameterSetCount )`
Function	The callback is called as soon as a Service Detail Response is received. Further details that are transmitted in this request can be queried with **SCC_GetServiceParameterSetData** **SCC_GetServiceParameterData** **SCC_GetServiceParameterNumericalValue** **SCC_GetServiceParameterPhysicalValue** **SCC_GetServiceParameterStringValue**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **ServiceId:** ID of the requested service (16 bit unsigned number). **ParameterSetCount:** Number of transmitted parameter sets.
Returns	-

ServicePayment-SelectionRes

Syntax	`void SCC_ServicePaymentSelectionRes ( byte SessionID[], long ResponseCode )`
Function	The callback is called as soon as a Service Payment Selection Response is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED”
Returns	-

PaymentDetailsRes

Syntax	`void SCC_PaymentDetailsRes ( byte SessionID[], long ResponseCode )`
Function	The callback is called as soon as a Service Payment Selection Response is received. Further details that are transmitted in this request can be queried with: **SCC_GetTimestamp** **SCC_GetGenChallenge** With `SCC_SetEnergyTransferType` the transfer type/mode sent in the following message can be set.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED”
Returns	-

Contract-AuthenticationRes

Syntax	`void SCC_ContractAuthenticationRes ( byte SessionID[], long ResponseCode )`
Function	The callback is called as soon as a Contract Authentication Response is received. Further details that are transmitted in this response can be queried with **SCC_GetProcessing**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED”
Returns	-

Certificate-InstallationRes

Syntax	`void SCC_CertificateInstallationRes ( byte SessionID[], long ResponseCode, char ContractID[] )`
Function	The callback is called as soon as a Certificate Installation Response is received. Further details that are transmitted in this response can be queried with **SCC_GetDHPublicKey** **SCC_GetEncryptedPrivateKey** **SCC_GetEMAIDIdAttr**.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **ContractID:** ID of the contract to which the certificate is assigned, max. 128 character string. (eMAID)
Returns	-

CertificateUpdateRes

Syntax	`void SCC_CertificateUpdateRes ( byte SessionID[], long ResponseCode, char ContractID[])`
Function	The callback is called as soon as a Certificate Installation Response is received. Further details that are transmitted in this response can be queried with **SCC_GetDHPublicKey** **SCC_GetEncryptedPrivateKey** **SCC_GetEMAIDIdAttr**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **ContractID:** ID of the contract to which the certificate is assigned, max. 128 character string. (eMAID)
Returns	-

ChargeParameter-DiscoveryRes

Syntax	`void SCC_ChargeParameterDiscoveryRes ( byte SessionID[], long ResponseCode, long SAScheduleTupleCount )`
Function	The callback is called as soon as a Charge Parameter Discovery Response is received. Further details that are transmitted in this request can be queried with the following functions: **SCC_GetMaxVoltage** **SCC_GetMaxCurrent** **SCC_GetMinCurrent** **SCC_GetSAScheduleTupleID** **SCC_GetProcessing** **SCC_GetMaxPower** (DC) **SCC_GetCurrentRegulationTolerance** (DC) **SCC_GetEnergyToBeDelivered** (DC) **SCC_GetPeakCurrentRipple** (DC) **SCC_GetMinVoltage** (DC) **SCC_GetNominalVoltage** (ISO 15118 AC)
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **SAScheduleTupleCount:** Number of transmitted tuples
Returns	-

PowerDeliveryRes

Syntax	`void SCC_PowerDeliveryRes ( byte SessionID[], long ResponseCode )`
Function	The callback is called as soon as a Power Delivery Response is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **EVSEID:** ID of charge point.
Returns	-

ChargingStatusRes

Syntax (AC)	`void SCC_ChargingStatusRes ( byte SessionID[], long ResponseCode, char EVSEID[], long SAScheduleTupleID, long ReceiptRequired )`
Function	The callback is called as soon as a Charging Status Response is received. Further details that are transmitted in this request can be queried with **SCC_GetMaxCurrent** **SCC_GetMeterInfoData**
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF **ResponseCode:** 1 if “OK”, 0 if “FAILED” **EVSEID:** ID of charge point. **SAScheduleTupleID:** ID of the selected SAScheduleTuple **ReceiptRequired:** Indicates if the vehicle is required to send a MeteringReceiptReq
Returns	-

MeteringReceiptRes

Syntax	`void SCC_MeteringReceiptRes ( byte SessionID[], long ResponseCode )`
Function	The callback is called as soon as a Metering Receipt Response is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED”
Returns	-

CableCheckRes

Syntax (DC)	`void SCC_CableCheckRes ( byte SessionID[], long ResponseCode )`
Function	The callback is called as soon as a Cable Check Response is received. Further details (EVSEProcessing) that are transmitted in this response can be queried with **SCC_GetProcessing**.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED”
Returns	-

PreChargeRes

Syntax (DC)	`void SCC_PreChargeRes ( byte SessionID[], long ResponseCode, float EVSEPresentVoltage )`
Function	The callback is called as soon as a PreCharge Response is received.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **EVSEPresentVoltage:** Present voltage value of charge point
Returns	-

CurrentDemandRes	**Syntax (DC)**	`void CurrentDemandRes ( byte SessionId[], long ResponseCode, float EVSEPresentVoltage, float EVSEPresentCurrent, byte LimitAchievedFlags[], char EVSEID[], long SAScheduleTupleID, long ReceiptRequired)`
	Function	The callback is called as soon as a Current Demand Response is received. Further details that are transmitted in this message can be queried with **SCC_GetMaxPower** **SCC_GetMaxVoltage** **SCC_GetMaxCurrent** **SCC_GetMeterInfoData** `(ISO 15118)`
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **EVSEPresentVoltage:** Present voltage value of charge point **EVSEPresentCurrent:** Present current of charge point **LimitAchievedFlags** (`ISO 15118`)**:** Array of three flags which correspond to current | voltage | power limit achieved, in this order **EVSEID** (`ISO 15118`)**:** ID of charge point. **SAScheduleTupleID** (`ISO 15118`)**:** ID of the selected SAScheduleTuple **ReceiptRequired** (`ISO 15118`)**:** Indicates if the vehicle is required to send a MeteringReceiptReq
	Returns	-

WeldingDetection-Res	**Syntax (DC)**	`void SCC_WeldingDetectionRes ( byte SessionID[], long ResponseCode, float EVSEPresentVoltage )`
	Function	The callback is called as soon as a Welding Detection Response is received.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED” **EVSEPresentVoltage:** Current voltage value of charge point
	Returns	-

SessionStopRes	**Syntax**	`void SCC_SessionStopRes ( byte SessionID[], long ResponseCode )`
	Function	The callback is called as soon as a Session Stop Response is received.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ResponseCode:** 1 if “OK”, 0 if “FAILED”
	Returns	-

5.3 Other Callback Functions

ErrorStateInd

Syntax	`void SCC_ErrorStateInd ( byte SessionID[], long ErrorState )`
Function	The callback is called as soon as the node enters or exits the error state.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **ErrorState:** 1 if the error state is entered, 0 if it is exited.
Returns	-

Note: The error state is active until the timer for a protocol restart expires.

RestartInd

Syntax	`void SCC_RestartInd ( byte SessionID[] )`
Function	The callback is called as soon as the vehicle restarts the protocol after a disconnection. Comment: With **SCC_GetRetriesLeft** the number of restarts remaining is queried.
Parameters	**SessionID:** 8-byte long SessionID of the **last** SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF.
Returns	-

Message Timeout

Syntax	`void SCC_V2G_EVCC_Msg_TimeoutInd ( byte SessionID[], dword& MessageID )`
Function	The callback is called as soon as an V2G_EVCC_Msg_Timeout occurs.
Parameters	**SessionID:** 8-byte long SessionID of the SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **MessageID:** Type of message to be sent (see section 9.1)
Returns	-

PreCharge Timeout

Syntax	`void SCC_V2G_EVCC_PreCharge_TimeoutInd ( byte SessionID )`
Function	The callback is called as soon as an V2G_EVCC_PreCharge_Timeout occurs.
Parameters	**SessionID:** 8-byte long SessionID of the SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF.
Returns	-

CableCheck Timeout

Syntax	`void SCC_V2G_EVCC_CableCheck_TimeoutInd ( byte SessionID[] )`
Function	The callback is called as soon as an V2G_EVCC_CableCheck_Timeout occurs.
Parameters	**SessionID:** 8-byte long SessionID of the SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF.
Returns	-

5.4 Auxiliary Functions for Setting of Request Details

Protocol flow

From the perspective of the vehicle, the possibility exists at several points in the protocol to intervene in the protocol sequence through a choice of parameters. For example, the choice can be made between AC and DC charging. You can control this selection by calling the following functions.

SetFaultNotification

Syntax	`void SCC_SetFaultNotification ( char FaultCode[], char FaultMsg[] )`
Function	Sets the fault code and message for the next V2G message. The fault code is contained within the V2G header.
Parameters	**FaultCode:** Desired fault code, which must be a valid enum value according to the specification **FaultMsg:** Desired fault message string. If this string is empty, the optional message element is omitted.
Returns	0: Not successful 1: Successful

SetServiceDetail-Request

Syntax	`long SCC_SetServiceDetailRequest ( dword ServiceId )`
Function	Requests the vehicle to send a Service Detail Request during the ServiceDiscoveryRes or ServiceDetailRes callback. It is possible for multiple Service Detail Requests, or none at all, to be sent. If this function is not called, the vehicle skips this message.
Parameters	**ServiceId:** ID of service that is to be requested, in hexadecimal representation.
Returns	1 if successful; otherwise 0

Note: This function must be called **during a callback before a Service Detail Request may be sent**, i.e. the callback for the messages Service Discovery Response or Service Detail Response.

SetEnergyTransferType

Syntax	`long SCC_SetEnergyTransferType ( char EnergyTransferType[] )` `long SCC_SetEnergyTransferType ( char EnergyTransferType[], dword Force )`
Function	Sets the desired charging mode **for a running SCC session**. For the list of all valid charging modes, see the SCC specification.
Parameters	**EnergyTransferType:** Desired charging mode as string. Please make sure the string entered is a valid according to the schema. **Force:** If set to 1, the vehicle is forced to set the charging mode even if the charge point hasn't offered it previously. If set to 0, another mode may be selected if this one is unavailable.
Returns	1 if successful; otherwise 0

SetPaymentOption

Syntax	`long SCC_SetPaymentOption ( char PaymentOption[], dword Force )`
Function	Sets the desired payment option **for a running SCC session**.
Parameters	**PaymentOption:** Desired payment option as string. For ISO 15118, the value must be either "Contract" (indicates PnC mode) or "ExternalPayment" (indicates EIM mode). **Force:** If set to 1, the vehicle is forced to set the payment option even if the charge point hasn't offered it previously. If set to 0, the payment option is only selected if it has been offered.
Returns	1 if successful; otherwise 0

Message parameters

The following functions can be used to affect various message sent by the vehicle.

Note: The functions need a running SCC session in order to work. Make sure to call them only after the Session Setup Request message is sent.

SetSelectedService

Syntax	`long SCC_SetSelectedService ( long ServiceID, long ParameterSetID )`
Function	Adds a service to the SelectedServiceList of the PaymentService-SelectionReq (former ServicePaymentSelectionReq) message.
Parameters	**ServiceID:** ID of the desired service **ParameterSetID:** ID of the desired parameter set (if set to 0, this optional parameter will be omitted)
Returns	1 if successful; otherwise 0

Note: For a correct message, use one of the ServiceIDs sent by the charge point in the message ServiceDiscoveryRes. The charge service is always selected automatically by the vehicle.

SetSelectedTariff-TableEntry

Syntax	`long SCC_SetSelectedScheduleTableEntry ( long ID )`
Function	Sets the ID of the selected SAScheduleTuple, which is sent in the messages PowerDeliveryReq and MeteringReceiptReq.
Parameters	**ID:** Desired SAScheduleTupleID
Returns	1 if successful; otherwise 0

Note: For a valid charge session, use one of the SAScheduleTupleIDs sent by the charge point in the message ChargeParameterDiscoveryRes. You can query them using the function `GetSAScheduleTupleID`. If no tuple is selected, the vehicle will automatically select the first or newest one.

SetMaxPower
SetMaxVoltage
SetMaxCurrent

Syntax	`long SCC_SetMaxPower ( float MaxPower )`
	`long SCC_SetMaxVoltage ( float MaxVoltage )`
	`long SCC_SetMaxCurrent ( float MaxCurrent )`

Function	Sets the limit for power / voltage / current. These limits are used in various messages for both AC and DC mode (the actual element name depends on the charging mode and the procotol version). Defaults can be set using the configuration file.
Parameters	**MaxPower / MaxVoltage / MaxCurrent:** Limit value to be set.
Returns	0: Not successful 1: Successful

SetMinCurrent

Syntax (AC)	`long SCC_SetMinCurrent ( float MinCurrent )`
Function	Sets the minimum current for the message ChargeParameterDiscoveryReq. A default value can be set using the configuration file.
Parameters	**MinCurrent:** Minimum current value to be set.
Returns	0: Not successful 1: Successful

SetTargetVoltage
SetTargetCurrent

Syntax (DC)	`long SCC_SetTargetVoltage ( float TargetVoltage )` `long SCC_SetTargetCurrent ( float TargetCurrent )`
Function	Sets the desired voltage / current. These limits are used in various DC messages. If no values are set, the vehicle will automatically calculate defaults.
Parameters	**TargetVoltage / TargetCurrent:** Value to be set.
Returns	0: Not successful 1: Successful

5.5 Auxiliary Functions for Querying of Response Details

EVSEStatus

The following functions query parts of the AC or DC EVSEStatus structure.

GetEVSEStatusCode

Syntax (DC)	`void SCC_GetEVSEStatusCode ( char EVSEStatusCode[] )`
Returns	Status code string of DC_EVSEStatus

GetEVSEIsolation-Status

Syntax (DC)	`void SCC_GetEVSEIsolationStatus ( char EVSEIsolationStatus )`
Returns	Isolations status from DC_EVSEStatus

GetRCD

Syntax (AC)	`long SCC_GetRCD ( )`
Returns	RCD flag from AC_EVSEStatus

GetEVSENotification

Syntax	`void SCC_GetEVSENotification ( char EVSENotification, long& NotificationMayDelay )`
Returns	EVSENotification and its maximum delay, if present, from DC_EVSEStatus

General queries

The following functions can be used for all response messages.

GetMsgHeaderFault Notification

Syntax	`long SCC_GetMsgHeaderFaultNotification ( char FaultCode[], char FaultMsg[] )`
Returns	0 if no fault code is contained in the V2G header 1 if a fault code is contained in the V2g header In the latter case, fault code and optionally fault message are copied to the output buffers. Missing fault message results in an empty string.

GetResponseCode-String

Syntax	`void SCC_GetResponseCodeString ( char ResponseCode[] )`
Returns	Queries the response code string (to the output buffer). This allows for evaluating its actual semantics, whereas each callback only returns a binary value indicating “OK” or “FAILED”.

Message related

The following functions query parts of specific messages.

SLAC_GetMSound-Target

Syntax	`void SCC_SLAC_GetMSoundTarget ( byte MSoundTarget[] )`
Function	Queries the target MAC Address for the M-Sounds (mandatory value: broadcast address).
Parameters	**MSoundTarget:** Buffer to which the address is written (6 byte).
Returns	-

SECCDiscoveryRes	**Syntax**	`void SCC_GetEVSEIP ( byte IpAddress[] )`
	Returns	IP (v6) transmitted from the charge point (16 byte, to the output buffer)
	Syntax	`long SCC_GetEVSEPort ( )`
	Returns	TCP port transmitted from the charge point.

SessionSetupRes	**Syntax**	`long SCC_GetTimestamp ( )`
	Returns	Current time stamp in UNIX format.

ServiceDiscoveryRes	**Syntax**	`void SCC_GetServiceData ( long Index, long& ServiceID, char ServiceName[], char ServiceType[], char ServiceScope[], long& FreeService )`
	Returns	ServiceName string (64 characters), ID, ServiceType, and ServiceScope string (32 characters) and “FreeService” flag with the specified list index < ServiceCount. When using a recent protocol version, an index of 0 refers to the element “ChargeService”.
	Syntax	`long SCC_GetPaymentOptions ( )`
	Returns	Available PaymentOptions, where 0 = ExternalPayment 1 = Contract 2 = both
	Returns	Flag “FreeService” of the service with the specified list index < ServiceCount. When using a recent protocol version, an index of 0 refers to the element “ChargeService”.
	Syntax (DIN 70121)	`long SCC_GetEnergyTransferType ( char EnergyTransferType[] )`
	Returns	Type of power supply, if present: AC_Charging = 0, DC_Charging = 1 The EnergyTransferType string is additionally written to the output buffer.
	Syntax (ISO 15118)	`long SCC_GetEnergyTransferModeCount ( )`
	Returns	The number of SupportedEnergyTransferModes.
	Syntax (ISO 15118)	`long SCC_GetEnergyTransferMode ( long Index, char EnergyTransferMode[] )`
	Returns	The EnergyTransferMode within the specified list index (to the output buffer).

ServiceDetailRes	**Syntax**	`void SCC_GetServiceParameterSetData ( long Index, long& ParameterSetID, long& ParameterCount )`
	Returns	ParameterSetID and number of parameters of the parameter set with the specified list index < ParameterSetCount (via reference).
	Syntax	`void SCC_GetServiceParameterData ( long i1, long i2, char Name[], char ValueType[] )`
	Returns	The name and value type of the parameter with the selected indices, where > i1 = index of the target ParameterSet > i2 = index of the target Parameter For DIN 70121, ValueType is the actual content of the element <ValueType>. For ISO 15118, the subelement holding the parameter value is evaluated and ValueType is set accordingly. In both cases, there are the same possible results for ValueType, e.g. “int”, “physicalValue”, “string”.
	Syntax	`long SCC_GetServiceParameterNumericalValue ( long i1, long i2 )`
	Returns	The numerical value of the parameter with the selected indices (see above). Use for value types “boolValue”, “byteValue”, “shortValue” or “intValue”.
	Syntax	`float SCC_GetServiceParameterPhysicalValue ( long i1, long i2 )`
	Returns	The physical value of the parameter with the selected indices (see above). Use for value type “physicalValue”.
	Syntax	`void SCC_GetServiceParameterStringValue ( long i1, long i2, char Value[] )`
	Returns	The string value of the parameter with the selected indices (see above), to the output buffer. Use for value type “stringValue”.

PaymentDetailsRes	**Syntax**	`void SCC_GetGenChallenge ( byte GenChallenge[] )`
	Returns	The challenge generated by the charge point (to a 16 byte output buffer).
	Syntax	`long SCC_GetTimestamp ( )`
	Returns	Current time stamp in UNIX format.

Certificate-InstallationRes and CertificateUpdateRes	**Syntax**	`void SCC_GetEncryptedPrivateKey ( byte Key[], char IdAttr[] )`
	Returns	Gets the encrypted private key (48 byte) of the new contract certificate and its Id attribute (to the output buffers).
	Syntax (ISO 15118)	`void SCC_GetEMAIDIdAttr ( char IdAttr[] )`
	Returns	Id attribute of the eMAID element

CertificateUpdateRes	**Syntax**	`long SCC_GetCertificateUpdateResRetryCounter ( )`
	Returns	In case of failure, this denotes when the EVCC should try to get the new Certificate again (number of days).

ChargeParameter-DiscoveryRes	**Syntax**	`float SCC_GetMaxVoltage ( )`
	Returns	Maximum line voltage of charge point (AC: “EVSEMaxVoltage”, DC: “EVSEMaximumVoltageLimit”).
	Syntax	`float SCC_GetMaxCurrent ( )`
	Returns	Maximum line current of charge point (AC: “EVSEMaxCurrent”, DC: “EVSEMaximumCurrentLimit”).
	Syntax	`float SCC_GetMinCurrent ( )`
	Returns	Minimum line current of vehicle (AC: “EVSEMinCurrent”, DC: “EVSEMinimumCurrentLimit”).
	Syntax (ISO 15118 AC)	`float SCC_GetNominalVoltage ( )`
	Returns	Supported line voltage of charge point
	Syntax (DC)	`float SCC_GetMaxPower ( )`
	Returns	Maximum power of charge point (AC: “EVSEMaxPower”, DC: “EVSEMaximumPowerLimit”).
	Syntax (DC)	`float SCC_GetCurrentRegulationTolerance ( )`
	Returns	Absolute value of regulation tolerance of charge point.
	Syntax (DC)	`float SCC_GetEnergyToBeDelivered ( )`
	Returns	Energy to be delivered by the charge point.
	Syntax (DC)	`float SCC_GetPeakCurrentRipple ( )`
	Returns	Peak-to-peak magnitude of the current ripple.
	Syntax (DC)	`float SCC_GetMinVoltage ( )`
	Returns	Minimum line voltage of charge point.
	Syntax	`long SCC_GetProcessing ( )`
	Returns	0 if “Finished”, 1 if “Ongoing” 2 if “Ongoing_WaitingForCustomerInteraction” (ISO 15118)
	Syntax	`long SCC_GetSAScheduleTupleID ( long Index )`
	Returns	ID of the SAScheduleTuple with the selected list index < TariffCount.

SAScheduleList	The following functions apply only to the SAScheduleList of ISO 15118:	
	Syntax	`long SCC_GetPMaxScheduleEntryCount ( long Index )`
	Returns	The number of PMaxScheduleEntries in the subelement PMaxSchedule of the SAScheduleTuple with the selected index.
	Syntax	`void SCC_GetPMaxScheduleEntryData ( long i1, long i2, long& Start, long& Duration, float& PMax )`

Returns	The start time, duration, and maximum power of the entry with the selected indices, where > i1 = index of the target SAScheduleTuple > i2 = index of the target PMaxScheduleEntry If the optional element “Duration” is not present, a value of -1 is returned.
Syntax	`void SCC_GetSalesTariffData ( long Index, char IdAttr[], long& SalesTariffId, char Description, long& NumEPriceLevels )`
Returns	ID attribute, SalesTariffId, Description and number of distinct price levels of the SalesTariff within the SAScheduleTuple with the selected index. If no SalesTariff is present, SalesTariffId is set to -1. This can be used to check the present of a SalesTariff in a SAScheduleTuple.
Syntax	`long SCC_GetSalesTariffEntryCount ( long Index )`
Returns	The number of SalesTariffEntries in the SalesTariff within the SAScheduleTuple with the selected index.
Syntax	`void SCC_GetSalesTariffEntryData ( long i1, long i2, long& Start, long& Duration, long& EPriceLevel, long& ConsumptionCostCount )`
Returns	The start time, duration, price level and the number of ConsumptionCost subelements of the SalesTariffEntry with the selected indices via references, where > i1 = index of the target SAScheduleTuple > i2 = index of the target SalesTariffEntry If the optional element “EPriceLevel” is not present, a value of -1 is returned.
Syntax	`void SCC_GetConsumptionCostData ( long i1, long i2, long i3, float& StartValue, long& CostCount )`
Returns	The start value and the number of Cost subelements of the ConsumptionCost element with the selected indices via references, where > i1 = index of the target SAScheduleTuple > i2 = index of the target SalesTariffEntry > i3 = index of the target ConsumptionCost element
Syntax	`void SCC_GetCostData ( long i1, long i2, long i3, long i4, char CostKind[], dword& Amount, long& AmountMultiplier, long& HasMultiplier )`
Returns	The kind, amount and multiplier (range [-3..3]) values of the Cost element with the selected indices via references, where > i1 = index of the target SAScheduleTuple > i2 = index of the target SalesTariffEntry > i3 = index of the target ConsumptionCost element > i4 = index of the target Cost element To denote that the multiplier is not present, the flag HasMultiplier is se to 0.

ChargingStatusRes
MeteringStatusRes

Syntax	`float SCC_GetMaxPower ( )`
Returns	Maximum power of charge point.

CableCheckRes

Syntax	`long SCC_GetProcessing ( )`
Returns	0 if “Finished”, 1 if “Ongoing” 2 if “Ongoing_WaitingForCustomerInteraction” (ISO 15118)

CurrentDemandRes

Syntax	`float SCC_GetMaxPower ( )`
Returns	Maximum power of charge point (“EVSEMaximumPowerLimit”).
Syntax	`float SCC_GetMaxVoltage ( )`
Returns	Maximum line voltage of charge point (“EVSEMaximumVoltageLimit”).
Syntax	`float SCC_GetMaxCurrent ( )`
Returns	Maximum line current of charge point (“EVSEMaximumCurrentLimit”).

Meter data

The following functions are available for the following messages, if a MeterInfo element is contained:

ChargingStatusRes
CurrentDemandRes
MeteringReceiptRes

Syntax	`void SCC_GetMeterInfoData ( char MeterID, float& MeterReading, byte SigMeterReading, long& MeterStatus, long& TMeter)`
Returns	Data assigned to the meter: ID (32 characters), current meter reading in [Wh], signature, status and timestamp in UNIX format Length of signature is dependent on the encryption algorithm.

5.6 Controlling the Vehicle Behavior

Configuration

The following functions change the vehicle configuration regarding the desired protocol version and encoding.

Reference: When setting schema namespace and version with the functions below, the behavior of the SupportedAppProtocolReq message also depends on the configuration parameter **PublishAllSchemas** (see 4.4).

SetBatteryState
SetBatterySOC

Syntax (DC)	`void SCC_SetBatteryState ( float BatteryState )` `void SCC_SetBatterySOC ( float BatteryState )`
Function	Sets the charging state of the battery, either in [Wh] with the first function or in [%] with the second. Note: The configured battery capacity cannot be overwritten.
Parameters	**BatteryState:** Desired charging state in Wh or percent.
Returns	-

SetChargeLoop-Interval

Syntax	`void SCC_SetChargeLoopInterval ( dword Interval )` `void SCC_SetChargeLoopInterval ( dword Interval, float jitterPercent )`
Function	Specifies the interval time between consecutive requests when inside the charge loop. Does not apply to MeteringReceipt.
Parameters	**Interval:** Desired interval time in milliseconds **JitterPercent**: Desired jitter (max. random variation) in % of Interval
Returns	-

SetWelding-DetectionCount

Syntax	`long SCC_SetWeldingDetectionCount ( dword Count )`
Function	Sets the number of WeldingDetection requests for the next welding detection phase.
Parameters	**Count:** Number of WeldingDetection messages, may be zero.
Returns	0: Not successful 1: Successful

EVStatus

The following functions change the statuses of the vehicle.

SetEVReady

Syntax	`long SCC_SetEVReady ( long Ready )`
Function	The function sets the ready flag.
Parameters	**Ready:** 1 = ready, 0 = not ready
Returns	0: Not successful 1: Successful

SetCabin-Conditioning

Syntax	`long SCC_SetCabinConditioning ( long CabinConditioning )`
Function	The function sets the cabin conditioning flag.
Parameters	**CabinConditioning:** 1 = conditioning on, 0 = conditioning off
Returns	0: Not successful 1: Successful

SetRESS-Conditioning

Syntax	`long SCC_SetRESSConditioning ( long RESSConditioning )`
Function	The function sets the RESS conditioning flag.
Parameters	**RESSConditioning:** 1 = conditioning on, 0 = conditioning off
Returns	0: Not successful 1: Successful

SetDCStatusCode

Syntax (DC)	`void SCC_SetDCStatusCode ( char StatusCode[] )`
Function	Sets the DC status.
Parameters	**StatusCode:** String buffer that is written to the status code.
Returns	-

Other functions

The following functions are not directly related to a specific message parameter.

Shutdown

Syntax	`long SCC_Shutdown ( long Terminate )`
Function	Stops the charging session when inside the charge loop, by starting a regular shutdown procedure.
Parameters	**Terminate:** If 1, the SessionStopReq is sent with ChargingSession = “Terminate”, else with ChargingSession = “Pause” (ISO 15118 only, else this value is ignored)
Returns	0: Not successful 1: Successful

InstantShutdown

Syntax	`long SCC_InstantShutdown ( long Terminate )`
Function	Stops the charging session immediately with a Session Stop request.
Parameters	**Terminate:** If 1, the SessionStopReq is sent with ChargingSession = “Terminate”, else with ChargingSession = “Pause” (ISO 15118 only, else this value is ignored)
Returns	0: Not successful 1: Successful

SuspendTx

Syntax	`void SCC_SuspendTx ( long NumberOfMessages )`
Function	Skips the sending of the following messages depending on the parameter value.
Parameters	**NumberOfMessages:**

Value	Behavior
-1	Sending of all further messages is suspended.
0	Resume. Starting from current state messages are send.
>0	The following “NumberOfMessages” are suspended.

Returns	-

StartRenegotiation

Syntax	`long SCC_StartRenegotiation ( )`
Function	Initiates a renegotiation procedure while inside the charge loop.
Parameters	**None**
Returns	0: Not successful 1: Successful

Note: The charge point may initiate a renegotiation procedure by sending an EVSENotification with the value “ReNegotiation” (see `SCC_SetEVSENotification`).

StartCertificate-Installation

Syntax	`long SCC_StartCertificateInstallation ( )`
Function	Schedules a CertificateInstallationReq message to be sent after the ServiceAndPaymentSelectionReq. This is only possible when in ISO 15118 PnC mode.
Parameters	**None**
Returns	0: Not successful 1: Successful

StartCertificate-Update

Syntax	`long SCC_StartCertificateUpdate ( )`
Function	Schedules a CertificateUpdateReq message to be sent after the ServiceAndPaymentSelectioReq. This is only possible when in ISO 15118 PnC mode.
Parameters	**None**
Returns	0: Not successful 1: Successful

5.7 Status Queries

SLAC_GetAtten-Results

Syntax	`void SCC_SLAC_GetAttenResults ( float Results[], long& ResultCount )`
Function	Queries the measured average attenuation values during the callback **SCC_SLACFinishedInd** (see section 6.1).
Parameters	**Results:** Array to which the attenuation results are written. The results are returned sorted, starting with the lowest attenuation. **ResultCount:** Variable to which the number of results in the array is written
Returns	-

GetSessionId

Syntax	`void SCC_GetSessionId ( byte SessionID [] )`
Function	Outputs the SessionID of the connection, or 0 if no connection exists.
Parameters	**SessionID:** 8-byte long array to which the SessionID is written.
Returns	-

GetDCStatusCode

Syntax (DC)	`void SCC_GetDCStatusCode ( char StatusCode[] )`
Function	Outputs the DC status in string form.
Parameters	**StatusCode:** String buffer to which the status code is written.
Returns	-

GetBatteryState
GetBatterySOC

Syntax (DC)	`float SCC_GetBatteryState ( )` `float SCC_GetBatterySOC ( )`
Function	Outputs the current charging state of the battery, either in [Wh] with the first function or in [%] with the second
Parameters	**None**
Returns	Charging state in Wh.

GetRetriesLeft

Syntax (DC)	`long SCC_GetRetriesLeft ( )`
Function	Specifies the remaining number of connection attempts.
Parameters	**None**
Returns	-

6 Shared CAPL Interface

In this chapter you will find the following information:

6.1 Callback Interface

Link Status Change

Syntax	`void SCC_LinkStatusChangeInd ( dword OldStatus, dword NewStatus )`
Function	The callback is called each time the internally stored link status changes by means of link status polling.
Parameters	**OldStatus:** Previous link status (0 = no link, 1 = link, 2 = unkown) **NewStatus:** New link status (0 = no link, 1 = link, 2 = unkown)
Returns	-

Note: The link status can also be changed manually using SCC_SetLinkStatus (see section 6.3)

State Transition

Syntax	`void SCC_StateTransitionInd ( byte SessionID[], dword OldState, dword NewState )`
Function	The callback is called each time the internal state machine switches its state, i.e. usually when a new message is received. Use **SCC_GetStateName** for a string representation of the state IDs (see section 6.5).
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **OldState:** Enum value of previous state **NewState:** Enum value of new state
Returns	-

Reference: For the meaning of the state values, see the appendix to this document (section 9.2).

Message Tx

Syntax	`void SCC_MessageTxInd ( byte SessionOrRunID[], dword MessageID, long Error )`
Function	The callback is called each time a Vehicle2Grid, SECC Discovery or SLAC message is sent by the simulation DLL.
Parameters	**SessionOrRunID:** 8-byte long SessionID (V2G) or RunID (SLAC) of the connection, range: 0 – 0xFF FF FF FF FF FF FF FF. **MessageID:** Type of sent message: - For SLAC messages, MMType (2 byte) according to specification - For V2G messages see section 9.1.4 **Error:** 0 if the sending was successful, > 0 if the send call failed (sending may fail due to socket errors or the absence of a receiver which sends ACK packages).
Returns	-

Invalid Message

Syntax	`void SCC_InvalidMessageInd ( long MessageType, long ErrorCode )`
Function	The callback is called when a SDP or V2G message is received with an invalid content (i.e. it is rejected by the session layer).
Parameters	**MessageType:** Type of the invalid message: 0: SDP (SECC Discovery Protocol) 1: V2G (Vehicle2Grid) **ErrorCode:** Type of error: 0: Wrong version number, or version number format 1: Unknown payload type 2: Invalid payload length 3: Error while decoding (EXI or XML parsing failed) – V2G only 4: Invalid signature – V2G only
Returns	-

Signature Verification

Syntax	`void SCC_SignatureVerificationInd ( dword MessageID, dword SignatureValid )`
Function	The callback is called when a signed V2G message is verified with the ECDSA algorithm.
Parameters	**MessageID:** Type of the received message (see section 9.1) **SignatureValid:** 1 if the signature was verified as valid, 0 if the signature was verified as invalid
Returns	-

Protocol Finished

Syntax	`void SCC_ProtocolFinishedInd ( byte SessionID[] )`
Function	The callback is called when a protocol run has been successfully finished.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF.
Returns	-

SLAC Finished

Syntax	`void SCC_SLACFinishedInd ( dword Result )`
Function	The callback is called when a SLAC run has been completed. It marks the point in time when the SECC Discovery process can be be started, if a match has been found and as soon as the link is established. The vehicle can use the function **SCC_SLAC_GetAttenResults** during this callback to get the measured attenuation values (see section **Fehler! Verweisquelle konnte nicht gefunden werden.**).
Parameters	**Result:** Result of the SLAC run (1 = match, 0 = no match)
Returns	-

Note: If you are not using an real QCA7000/7005 chip, use `SCC_SetLinkStatus(1)` during this callback to make the SECC Discovery start instantly.

Connection Timeout

Syntax	`void SCC_ConnectionTimeoutInd ( byte SessionID[] )`
Function	The callback is called when a connection is closed due to inactivity.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 – 0xFF FF FF FF FF FF FF FF.
Returns	-

TCP Connect

Syntax	`void SCC_TCPConnectInd ( dword Port, long IsInitiator )`
Function	The callback is called when a TCP connection is opened.
Parameters	**Port:** Port number of the TCP connection. **IsInitiator:** 1 if the node layer initiated the connection (client case), 0 if the node layer received the connection request (server case)
Returns	-

TCP Shutdown

Syntax	`void SCC_TCPShutdownInd ( dword Port, long IsInitiator )`
Function	The callback is called when a TCP connection is closed.
Parameters	**Port:** Port number of the TCP connection. **IsInitiator:** 1 if the node layer initiated the shutdown, 0 if the node layer received the shutdown request
Returns	-

6.2 Auxiliary Functions for Querying of Message Details

Note: The functions can be called only within the assigned callback. They are used to query the details of a SLAC or V2G message.

GetMessageRxTime

Syntax	`qword SCC_GetMessageRxTime ( )` `void SCC_GetMessageRxTime ( dword& TimeNsHigh, dword& TimeNsLow )`
Function	Queries the CANoe internal timestamp of the message receipt event, i.e. the CANoe simulation time, via references. The function is available for all kinds of message callbacks.
Parameters	-
Returns	Timestamp in nanoseconds

Note: The result of this query is the message timestamp as seen in the Trace window. It is a value taken from CANoe, in constract to **SCC_GetTimestamp**, which returns a timestamp parameter that is sent inside a V2G message.

GetVerificationStatus

Syntax	`dword SCC_GetVerificationStatus ( )`
Function	Returns the state of the received message regarding the validity of its signature.
Parameters	-
Returns	0 = Verification failed 1 = Verification successful 2 = Not verified (unsigned message or not required)

GetHeaderData

Syntax	`void SCC_GetHeaderData ( byte HeaderData[] )`
Function	Returns the SDP or V2G message header (to the output buffer).
Parameters	**HeaderData:** 8 byte output buffer
Returns	V2G / SDP Transport Layer message header

SLAC_GetEth-PayloadLength

Syntax	`dword SLAC_GetEthPayloadLength ( )`
Function	Returns the Ethernet payload of a SLAC message. (Does not work with SDP or V2G frames)
Parameters	-
Returns	Ethernet payload length

SLAC_Get-DestinationAddress

Syntax	`long SCC_SLAC_GetDestinationAddress ( byte[] MACAddress )`
Function	Returns the destination MAC address of a SLAC message, to the output buffer
Parameters	**MACAddress:** Destination address of the MME frame
Returns	-

Note: The destination address may differ from the simulation node's address when the node is in passive mode. This function is intended to allow filtering for the node's address in this case.

SLAC_Get-ApplicationType

Syntax	`long SCC_SLAC_GetApplicationType ( )`
Function	Queries the application type.
Parameters	-
Returns	0x00 = PEV-EVSE Association (mandatory) 0x01-0xFF = Reserved

SLAC_GetSecurity-Type

Syntax	`long SCC_SLAC_GetSecurityType ( )`
Function	Queries the security type.
Parameters	-
Returns	0x00 = No Security (mandatory) 0x01 = Public Key Signature 0x02-0xFF = Reserved

SLAC_GetRespType

Syntax	`long SCC_SLAC_GetRespType ( )`
Function	Queries the response type.
Parameters	-
Returns	0x00 = HLE of the STA 0x01 = Another GP STA (mandatory) 0x02-0xFF = Reserved

SLAC_GetSignal-Type

Syntax	`long SCC_SLAC_GetSignalType ( )`
Function	Queries the signal type.
Parameters	-
Returns	0x00 = PEV S2 toggles on CPLT line (mandatory) 0x01-0xFF = Reserved

SLAC_GetSourceId

Syntax	`void SCC_SLAC_GetSourceId ( byte SourceID[] )`
Function	Queries the Source ID.
Parameters	**SourceID:** Buffer to which the ID is written (17 byte).
Returns	-

SLAC_Get-ResponseId

Syntax	`void SCC_SLAC_GetResponseId ( byte ResponseID[] )`
Function	Queries the Response ID.
Parameters	ResponseID: Buffer to which the ID is written (17 byte).
Returns	-

SLAC_GetPEVAnd-EVSEId

Syntax	`void SCC_SLAC_GetPEVAndEVSEId ( byte PEVID[], byte EVSEID )`
Function	Queries the PEV and EVSE IDs.
Parameters	**PEVID:** Buffer to which the PEV ID is written (17 byte). **EVSEID:** Buffer to which the EVSE ID is written (17 byte).
Returns	-

SLAC_GetMVF-Length

Syntax	`long SCC_SLAC_GetMVFLength ( )`
Function	Queries the length of the match variable field.
Parameters	-
Returns	Length of the field

SLAC_GetReservedField

Syntax	`long SCC_SLAC_GetReservedField ( dword Index, byte Data[], dword& DataSize )`
Function	Queries one of the reserved fields of the message. For a valid message, these fields must contain only zeroes.
Parameters	**Index:** Number of the reserved field (0 or 1) **Data:** Output buffer for the field **DataSize:** Output of the byte length of the returned data
Returns	Data and length of the reserved field.

SLAC_GetCM-SetKeyCnfData

Syntax	`void SLAC_GetCMSetKeyCnfData ( dword& MyNonce, dword& YourNonce, dword& PID, dword& PRN, dword& PMN, dword& CCOCapability )`
Function	Queries the additional parameters of a CM_Set_Key.Cnf message, all at once, via references.
Parameters	**MyNonce:** Random number that will be used to verify next message (usually 0) **YourNonce:** Last nonce received; used to verify this message (usually 0) **PID:** Protocol (usually 0x04) **PRN:** Protocol Run Number (usually 0) **PMN:** Protocol Message Number (usually 0) **CCOCapability:** STA's CCo capability (usually 0)
Returns	-

GetCertificateChain-Data

Syntax	`void SCC_GetCertificateChainData ( long Target, char IdAttr[], long& SubCertificateCount )`
Function	Reads Id (to the output buffer) and number of sub certificates (via reference) of the target certificate chain. The ID is only available for `ISO 15118`.
Parameters	**Target:** set according to the type of certificate chain that is queried: 0 = ContractSignatureCertChain 1 = SAProvisioningCertChain (EV and `ISO 15118` only)
Returns	-

GetCertificateChain-Certificate

Syntax	`void SCC_GetCertificateChainCertificate ( long Target, long Index, char Certificate[] )`
Function	Reads a certificate from the target certificate chain.
Parameters	**Index:** Index of the target certificate, where 0 denoted the parent certificate, and 1..4 denote sub-certificates **Target:** Set this according to the type of certificate chain that is queried: 0 = ContractSignatureCertChain 1 = SAProvisioningCertChain (EV and `ISO 15118` only) **Certificate:** Target certificate as base64 string
Returns	-

GetDHPublicKey

Syntax	`void SCC_GetDHPublicKey ( byte Key[], char IdAttr[] )`
Function	Reads the Diffie-Hellman parameter (publiy key) from a Certificate Installation/Update Response or a a Certificate Installation/Update Request (the latter only for versions other than `ISO 15118`).
Parameters	**Key:** Value of element DHPublicKey (`ISO 15118`) resp. DHParams (other versions) **IdAttr:** Id attribute of DHPublicKey (`ISO 15118` only)
Returns	-

6.3 Controlling the Simulation State

Simulation State

The Nodelayer DLLs can take four different states (Off, Passive, Active and Paused), which are controlled by the following functions.

StartSimulation

Syntax	`long SCC_StartSimulation ( )` `long SCC_StartSimulation ( dword SLACMode )`
Function	These functions start the simulation DLL in active mode. **Until the simulation is started, no messages are sent and the API (except `SCC_StartSimulation()`) has no effect!** With the vehicle DLL, the call of `SCC_StartSimulation()` starts the setup of a connection.
Parameters	**SLACMode:** Controls the behavior regarding SLAC, which overrides the parameter <UseSLAC> from the XML configuration: 0 = SLAC is skipped (may be done manually) 1 = SLAC is used 2 = EV: use SLAC depending on link status (not recommended due to QCA chip instabilities)
Returns	0: Not successful 1: Successful

Note: If simulation is restarted after a stop, the XML configuration file will be read in again. While the simulation is deactivated, another data set can be loaded with `LoadCommunicationConfig` (see below).

StartPassive

Syntax	`long SCC_StartPassive ( )` `long SCC_StartPassive ( dword SLACMode )`
Function	This function activates the Nodelayer DLL, but does not start an SCC simulation (i.e. a state machine). In this state, callbacks can be received, and test functions can be called (see chapter 7).
Parameters	**SLACMode:** Controls the behavior regarding SLAC, which overrides the parameter <UseSLAC> from the XML configuration: 0 = SLAC is skipped (may be done manually) 1 = SLAC is used 2 = EV: use SLAC depending on link status (not recommended due to QCA chip instabilities)
Returns	0: Not successful 1: Successful

StopSimulation

Syntax	`long SCC_StopSimulation ( )`
Function	These functions activate/deactivate the SCC simulation. **If the simulation is deactivated, no messages are sent and the API (except `SCC_StartSimulation()`) has no effect!** With the vehicle DLL, the call of `SCC_StartSimulation()` starts the setup of a connection.
Parameters	-
Returns	0: Not successful 1: Successful

Note: If simulation is restarted after a stop, the XML configuration file will be read in again. While the simulation is deactivated, another data set can be loaded with `LoadCommunicationConfig` (see below).

SimulationWait
SimulationResume

Syntax	`long SCC_SimulationWait ( )` `long SCC_SimulationResume ( )`
Function	These functions pause/resume the SCC simulation. This functionality can be used to freeze the protocol flow in a certain state.
Parameters	-
Returns	0: Not successful 1: Successful

Reference: For further information on this feature and its usage, see CANoe help.

SimulationReset

Syntax	`long SCC_SimulationReset ( )`
Function	This function resets the SCC simulation to its initial state, corresponding to the start of a new measurement. All connections and stored parameters are deleted.
Parameters	-
Returns	0: Not successful 1: Successful

GetSimulationState

Syntax	`long SCC_GetSimulationState ( )`
Function	This function checks if the simulation is running, waiting (in “pause” state) or stopped.
Parameters	-
Returns	0: Simulation is deactivated 1: Simulation is running in passive mode 2: Simulation is running in active mode 3: Simulation is running in active mode and waiting (`SCC_SimulationWait()` has been called)

SLAC_GetLinkStatus		
	Syntax	`long SCC_SLAC_GetLinkStatus ( )`
	Function	This function retrieves the internally stored link status of the QCA7000/7005 chip.
	Parameters	-
	Returns	0: No link available 1: Link is available 2: Link status is unkown (chip is not responding)

SLAC_SetLinkStatus		
	Syntax	`long SCC_SLAC_SetLinkStatus ( long LinkStatus )`
	Function	This function overrides the internally stored link status of the QCA7000/7005 chip. This is useful if no hardware is present.
	Parameters	**LinkStatus:** Value to be set: 0: No link available 1: Link is available 2: Link status is unkown (chip is not responding)
	Returns	0: Not successful 1: Successful

SLAC_SetLink-StatusPollingType		
	Syntax	`long SCC_SLAC_SetLinkStatusPollingType ( dword Type )`
	Function	Changes the message(s) used for querying the link status (if activated). This overwrites the configuration parameter <SLAC_LinkStatusPollingType>.
	Parameters	**Type:** Desired message configuration, where 0 = use both available messages 1 = use only VS_PL_Lnk_Status 2 = use only VS_Nw_Info
	Returns	0: Not successful 1: Successful

SLAC_SetLink-StatusPollingInterval		
	Syntax	`long SCC_SLAC_SetLinkStatusPollingInterval ( long Interval )`
	Function	This function changes the polling interval for sending VS_PL_Lnk_Status or VS_Nw_Info requests to the chip. Set the interval to 0 to disable polling.
	Parameters	**Interval:** Desired time interval in milliseconds
	Returns	0: Not successful 1: Successful

6.4 Configuration and Behavior

The following functions change the configuration of the simulation.

Loading of XML configuration

Restricted to certain protocol states!

Syntax	`long SCC_LoadCommunicationConfig ( long ConfigID )` `long SCC_LoadV2GConfig ( long ConfigID )`
Function	Loads the section within the XML configuration file designated with `configID`. If `LoadCommunicationConfig` has not yet been called when the simulation starts, section 0 is loaded automatically. The function `SCC_LoadCommunicationConfig` loads the configuration globally, i.e. for all connections to be created. This call is only possible when simulation is deactivated. The function `SCC_LoadV2GConfig` loads the configuration for an already active connection, **applying only those parameters relevant to an individual connection**. This function can only be used within a callback; the configuration is then applied to the connection that has triggered the callback,
Parameters	**ConfigID:** ID of desired configuration section
Returns	0: Not successful 1: Successful

Note: The charge point DLL uses the <EVSEConfiguration> XML elements, and the vehicle DLL the <PEVConfiguration> elements. The two configuration types can be numbered independently.

Note: Using `SCC_LoadV2GConfig()`, the charge point may react on the requested schema, by supplying e.g. different types of tariff tables and service lists.

SetTLSEnabled

Syntax	`void SCC_SetTLSEnabled ( long Enabled )`
Function	Specifies if TLS is used. As native TLS is not supported, this only affects the “Security” flag in the SECC Discovery Request message. By calling this function, the configuration parameter <UseTLS> is overwritten.
Parameters	**Enabled:** 1 to enable TLS support, 0 to disable TLS support
	-

SetSchema

Syntax	`void SCC_SetSchema ( char Namespace[] )` `void SCC_SetSchema ( char Namespace[], long VersionMajor, long VersionMinor )`
Function	Sets the preferred schema version, corresponding to <SchemaNamespace> and optionally a specific version, in the XML configuration file. The configured schema will be be used unless a different one is derived from the SupportedAppProtocol handshake, and it will be priorized during the handshake. If unspecified, the latest schema will have the highest priority. When no version numbers are specified, depending on the context, it will mean “any” or “latest” version.
Parameters	**SchemaNamespace:** Desired namespace **SchemaVersionMajor:** Desired major version number, may be -1 for EVSE simulation **SchemaVersionMinor:** Desired minor version number, may be -1 for EVSE simulation
Returns	0: Not successful 1: Successful

GetEthernetSettings

Syntax	`void SCC_GetEthernetSettings(byte MacAddress[], byte IPv4Address[], dword& IPv4Available, byte IPv6Address[], dword& IPv6Available, long UDPPort, long TCPPort)`
Function	Retrieves the configured addresses and ports for the SCC node. These values may originate from the CANoe configuration or from the DLL’s XML configuration
Parameters	**MacAddress:** 6 byte MAC address **IPv4Address:** 4 byte IPv4 address **IPv4Available:** 1 if IPv4Address is valid, else 0 **IPv6Address:** 16 byte IPv6 address **IPv6Available:** 1 if IPv6Address is valid, else 0 **UDPPort:** UDP port number for SECC Discovery **TCPPort:** TCP port number for Vehicle2Grid TP
Returns	-

6.5 Other CAPL Functions

GetDLLInfo

Syntax	`dword SCC_GetDLLInfo ( dword selector )`
Function	This function returns additional information to the DLL.
Parameters	**selector:** Selects the desired return value (see below).
Returns	

selector	Description of return value
0	The file version of the DLL in BCD format, e.g., 0x010203 for V1.2.3
1	The main version of the DLL
2	The subversion of the DLL
3	The build number of the DLL

SetVerbosity

Syntax	`void SCC_SetVerbosity ( dword Level )`
Function	Sets the “Verbosity” parameter of the DLL. The higher the value is, the more information will be output in the write window.
Parameters	**Level:** The desired verbosity level.
Returns	-

Level	Behavior
0	Signals only critical errors
1	Signals errors due to erroneous configurations, etc.
2	Warns, if an unexpected protocol status is reached, and signals missing obligatory callbacks
3	Informs about missing optional callbacks and minor problems
>3	Signals low level events such as setting and expiring of timers

SetMessageDelay

Syntax	`void SCC_SetMessageDelay ( dword Delay )` `void SCC_SetMessageDelay ( dword Delay, float JitterPercent )`
Function	Specifies the delay time before an SCC message is sent. This allows you to slow down the protocol sequence. The delay applies to all active connections of the module.
Parameters	**Delay:** Desired delay value in milliseconds **JitterPercent**: Desired jitter (max. random variation) in % of Delay
Returns	-

GenerateRandom-Data

Syntax	`void SCC_GenerateRandomData ( byte Buffer[], long BufferLength )`
Function	Fills a buffer with random data, e.g. for use with the test functions.
Parameters	**Buffer:** Target buffer to which the output is written **BufferLength:** Number of bytes to fill
Returns	-

SLAC_GenerateNID

Syntax	`void SCC_SLAC_GenerateNID ( byte NMK[], byte NID_out[] )`
Function	Generates a matching NID to a given NMK, using the specified algorithm, to the output buffer. Doesn't affect the internal state.
Parameters	**NMK:** Network Membership Key (16 byte) **NID_out:** Network ID (7 byte hexadecimal number)
Returns	-

GetStateName

Syntax	`void SCC_GetStateName ( dword State, char Buffer[], int BufferLength )`
Function	Gets a string representation of a state enum value obtained with SCC_StateTransitionInd (see section 6.1).
Parameters	**State:** State enum to be translated **Buffer:** Target buffer to which the output is written **BufferLength:** Size of the output buffer
Returns	-

7 Test Functions

In this chapter you find the following information:

7.1 Overview

Test API

Test functions provide means to influence the SCC protocol outside the bounds of the regular protocol flow. The functions are always available, indifferent to the protocol's state machine, or if a protocol instance is even running.

Reference: See the chapter "Test Functions" in the CANoe help for details about the usage and configuration of the test functions.

7.2 Creation of SLAC messages

CreateCM_SLAC_Parm_Req

Syntax	`void SCC_CreateCM_SLAC_Parm_Req ( byte RunId[], byte SourceMac[], byte TargetMac[] )`
Function	Creates a CM_Slac_Parm.Req message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame
Returns	-

Additional parameters

Index	Name	Type	Description
0	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
1	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved

CreateCM_SLAC_Parm_Cnf

Syntax	`void SCC_CreateCM_SLAC_Parm_Cnf ( byte RunId[], byte SourceMac[], byte TargetMac[], dword NumSounds, dword TimeOut, byte ForwardingSTA[] )`
Function	Creates a CM_SLAC_Parm.Cnf message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **NumSounds:** Number of M-Sounds to be transmitted **TimeOut:** Timeout for transmission of M-Sounds in multiples of 100ms **ForwardingSTA:** MAC address where the measurement results shall be sent to
Returns	-

Additional parameters

Index	Name	Type	Description
0	RespType	dword	0x00 = HLE of the STA 0x01 = Another GP STA 0x02-0xFF = Reserved
1	MSoundTarget	byte[]	Target MAC address for M-Sounds
2	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
3	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved

CreateCM_Start_Atten_Char_Ind

Syntax	`void SCC_CreateCM_Start_Atten_Char_Ind ( byte RunId[], byte SourceMac[], byte TargetMac[], dword NumSounds, dword TimeOut, byte ForwardingSTA[] )`
Function	Creates a CM_Start_Atten_Char.Ind message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **NumSounds:** Number of M-Sounds transmitted during the SLAC process **TimeOut:** Timeout for transmission of M-Sounds in multiples of 100ms **ForwardingSTA:** MAC address where the measurement results shall be sent to
Returns	-

Additional parameters

Index	Name	Type	Description
0	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
1	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved
2	RespType	dword	0x00 = HLE of the STA 0x01 = Another GP STA 0x02-0xFF = Reserved

CreateCM_MNBC_Sound_Ind

Syntax	`void SCC_CreateCM_MNBC_Sound_Ind ( byte RunId[], byte SourceMac[], byte TargetMac[], dword Count )`
Function	Creates a CM_MNBC_Sound.Ind message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMacAddress:** MAC address of sender **Count:** Countdown counter for number of Sounds remaining
Returns	-

Additional parameters

Index	Name	Type	Description
0	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
1	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved
2	SenderId	byte[]	17 byte ID
3	Reserved field	byte[]	8 byte
4	Random number	byte[]	16 byte, if not set a random value will be generated

CreateCM_Atten_Profile_Ind

Syntax	`void SCC_CreateCM_Atten_Profile_Ind ( byte SourceMac[], byte TargetMac[], byte PEVMAC[], dword NumGroups, byte AAG[] )`
Function	Creates a CM_Atten_Profile.Ind message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **PEVMAC:** PEV MAC Address **NumGroups:** Number of attenuation groups **AAG:** Average Attenuation Group (array length is indicated by the paremeter 'NumGroups')
Returns	-

Additional parameters

Index	Name	Type	Description
0	Reserved field	dword	1 byte

CreateCM_Atten_Char_Ind

Syntax	`void SCC_CreateCM_Atten_Char_Ind ( byte RunId[], byte SourceMac[], byte TargetMac[], byte SourceAddress[], dword NumSounds, dword NumGroups, byte AAG[] )`
Function	Creates a CM_Atten_Char.Ind message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **SourceAddress:** MAC Address of the PEV which initiates the SLAC process **NumSounds:** Number of M-Sounds used to generate the ATTEN_PROFILE **NumGroups:** Number of attenuation groups **AAG:** Average Attenuation Group (array length is indicated by the paremeter 'NumGroups')
Returns	-

Additional parameters

Index	Name	Type	Description
0	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
1	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved
2	SourceId	byte[]	17 byte ID
3	RespId	byte[]	17 byte ID

CreateCM_Atten_Char_Rsp

Syntax	`void SCC_CreateCM_Atten_Char_Rsp ( byte RunId[], byte SourceMac[], byte TargetMac[], byte SourceAddress[] )`
Function	Creates a CM_Atten_Char.Rsp message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **SourceAddress:** MAC Address of the PEV which initiates the SLAC process
Returns	-

Additional parameters

Index	Name	Type	Description
0	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
1	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved
2	Result	dword	0x00 = Success
3	SourceId	byte[]	17 byte ID
4	RespId	byte[]	17 byte ID

CreateCM_Validate_Req

Syntax	`void SCC_CreateCM_Validate_Req ( byte SourceMac[], byte TargetMac[], dword ListenTimer )`
Function	Creates a CM_Validate.Req message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **ListenTimer:** Time duration while the EVSE shall listen to BCB-toggles: 0x00 = 100 ms 0x01 = 200 ms
Returns	-

Additional parameters

Index	Name	Type	Description
0	SignalType	dword	0x00 = PEV S2 toggles on CPLT line 0x01-0xFF = Reserved
1	Result	dword	0x00 = Not Ready 0x01 = Ready 0x02 = Success 0x03 = Failure 0x04 = Not Required 0x05-0xFF = Reserved

CreateCM_Validate_Cnf

Syntax	`void SCC_CreateCM_Validate_Cnf ( byte SourceMac[], byte TargetMac[], dword Result, dword ToggleNum )`
Function	Creates a CM_Validate.Cnf message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **Result:** Result code: 0x00 = Not Ready 0x01 = Ready 0x02 = Success 0x03 = Failure 0x04 = Not Required 0x05-0xFF = Reserved **ToggleNum:** Number of BC-edges detected by the EVSE
Returns	-

Additional parameters

Index	Name	Type	Description
0	SignalType	dword	0x00 = PEV S2 toggles on CPLT line 0x01-0xFF = Reserved

CreateCM_SLAC_Match_Req

Syntax	`void SCC_CreateCM_SLAC_Match_Req ( byte RunId[], byte SourceMac[], byte TargetMac[], byte PEVMacAddress[], byte EVSEMacAddress[] )`
Function	Creates a CM_SLAC_Match.Req message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **PEVMacAddress:** MAC Address of the PEV **EVSEMacAddress:** MAC Address of the EVSE
Returns	-

Additional parameters

Index	Name	Type	Description
0	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
1	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved
2	MVFLength	dword	Length of the match variable field
3	PEVID	byte[]	17 byte ID
4	EVSEID	byte[]	17 byte ID
5	Reserved field	byte[]	8 byte

CreateCM_SLAC_Match_Cnf

Syntax	`void SCC_CreateCM_SLAC_Match_Cnf ( byte RunId[], byte SourceMac[], byte TargetMac[], byte PEVMAC[], byte EVSEMAC[], byte NID[], byte NMK[] )`
Function	Creates a CM_SLAC_Match.Cnf message for sending.
Parameters	**RunId:** Random Run Identifier of sender (8 byte) **SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **PEVMAC:** PEV MAC Address **EVSEMAC:** EVSE MAC Address **NID:** Network ID given by the CCo (EVSE) (7 byte) **NMK:** Private NMK of the EVSE (random 16 byte value)
Returns	-

Additional parameters

Index	Name	Type	Description
0	ApplicationType	dword	0x00 = PEV-EVSE Association 0x01-0xFF = Reserved
1	SecurityType	dword	0x00 = No Security 0x01 = Public Key Signature 0x02-0xFF = Reserved
2	MVFLength	dword	Length of the match variable field
3	PEVID	byte[]	17 byte ID
4	EVSEID	byte[]	17 byte ID
5	Reserved field	byte[]	8 byte
6	Reserved field	dword	1 byte

CreateCM_Set_Key_Req

Syntax	`void SCC_CreateCM_Set_Key_Req ( byte SourceMac[], byte TargetMac[], byte NID[], byte NMK[] )`
Function	Creates a CM_Set_Key.Req message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **NID:** Network ID given by the CCo (EVSE) (7 byte) **NMK:** NMK to be set (16 byte)
Returns	-

Additional parameters

Index	Name	Type	Description
0	KeyType	dword	0x00 = DAK (AES-128) 0x01 = NMK (AES-128) 0x02 = NEK (AES-128) 0x03 = TEK (AES-128) 0x04 = Hash Key (Random-3072) 0x05 = Nonce Only (no key) 0x06-0xFF = Reserved
1	MyNonce	byte[]	4 byte random number
2	YourNonce	byte[]	Last receveived nonce (4 byte)
3	PID	dword	Protocol ID
4	PRN	byte[]	Protocol Run Number (2 byte)
5	PMN	dword	Protocol Message Number
6	CCoCapability	dword	STA's CCo Capability
7	NewEKS	dword	PEKS or EKS

CreateCM_Set_Key_Cnf

Syntax	`void SCC_CreateCM_Set_Key_Cnf ( byte SourceMac[], byte TargetMac[], dword Result )`
Function	Creates a CM_Set_Key.Cnf message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **Result:** Result code: 0x00 = success 0x01 = failure 0x02-0xFF = reserved
Returns	-

Additional parameters

Index	Name	Type	Description
0	MyNonce	byte[]	4 byte random number
1	YourNonce	byte[]	Last receveived nonce (4 byte)
2	PID	dword	Protocol ID
3	PRN	byte[]	Protocol Run Number (2 byte)
4	PMN	dword	Protocol Message Number
5	CCoCapability	dword	STA's CCo Capability

CreateVS_PL_Status_Req

Syntax	`void SCC_CreateVS_PL_Lnk_Status_Req ( byte SourceMac[], byte TargetMac[] )`
Function	Creates a VS_PL_Lnk_Status.Req message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame
Returns	-

Additional parameters

Index	Name	Type	Description
0	OUI	byte[]	3 byte Organizationally Unique Identifier

CreateVS_Nw_Info_Req

Syntax	`void SCC_CreateVS_Nw_Info_Req ( byte SourceMac[], byte TargetMac[] )`
Function	Creates a VS_Nw_Info.Req message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame
Returns	-

CreateVS_PL_Status_Cnf

Syntax	`void SCC_CreateVS_PL_Lnk_Status_Cnf ( byte SourceMac[], byte TargetMac[], dword MStatus, dword LinkStatus )`
Function	Creates a VS_PL_Lnk_Status.Cnf message for sending.
Parameters	**SourceMac:** Source address of the Ethernet frame **TargetMac:** Destination address of the Ethernet frame **MStatus:** MME Status: 0x00 = success 0xFF = failure **LinkStatus:** Indicates if a link is established: 0x00 = no link 0x01 = link
Returns	-

Additional parameters

Index	Name	Type	Description
0	OUI	byte[]	3 byte Organizationally Unique Identifier

7.3 Additional Parameters and Sending

The following functions are needed for modification and sending of the message created using the functions of section 7.2.

SLAC_SetAdditional-Parameter

Syntax	`long SCC_SLAC_SetAdditionalParameter ( dword Parameter, dword ParameterValue )` `long SCC_SLAC_SetAdditionalParameter ( dword Parameter, byte ParameterValues[] )`
Function	Sets an additional parameter after creating a message, thus overwriting a value predefined by the specification (DIN 70121:2014-12).
Parameters	**Parameter:** ID number of the parameter, as detoned in the lists of optional parameters below each Create-function **ParameterValue:** Desired value of the parameter in the matching data type
Returns	1 if successful, 0 else

Note: Please make sure to use the correct value type and array length for the respective parameter. The parameter types are detoned in the lists of optional parameters below each Create-function.

SendPrepared-Message

Syntax	`long SCC_SendPreparedMessage ( )`
Function	Sends a message created using one of the Create()-functions.
Parameters	-
Returns	1 if successful, 0 else

Note: If multiple Create-functions are called without using `SCC_SendPreparedMessage()` in between, all prior messages are discarded. Only the last created message is available for sending.

7.4 Configuration Parameters

Message parameters

The following parameters are used as input data for one or more messages. Here, they are denoted by their top-level XML element only. For the detailed structure of their subelements, please refer to the ISO 15118 specification. In order to successfully send the corresponding message, all mandatory subelements must be specified!

Usage of certificates

Certificates are referred by their Name property as displayed in the Vector Security Manager after importing the certificate.

Note: Only leaf certificates are referred in the XML config. Please make sure that the base certificates are also part of the Security Profile where full chains are required, as they will be resolved automatically using the Signer ID.

Reference: For the usage of the Vector Security Manager, please refer to the CANoe help.

Name	Opt.[1]	Mult.[2]	Related message
AppProtocol		X	SupportedAppProtocolReq
SelectedServiceList[3]			ServicePaymentSelectionReq
ContractCert			PaymentDetailsReq, CertificateInstallationRes, CertificateUpdateReq CertificateUpdateRes
ContractCertPrivateKey			CertificateInstallationRes, CertificateUpdateRes
MOSub2Cert			ChargeParameter-DiscoveryRes
OEMProvisioningCert			CertificateInstallationReq
SAProvisioningCert			CertificateInstallationRes, CertificateUpdateRes
ListOfRootCertificateIDs			CertificateInstallationReq, CertificateUpdateReq
ChargingProfile	X		PowerDeliveryReq
PaymentOptions / PaymentOptionList (ISO 15118)			ServiceDiscoveryRes
ChargeService			
ServiceList	X		
ServiceParameterList	X		ServiceDetailRes
SAScheduleList			ChargeParameter-DiscoveryRes

[1] Optional element

[2] Multiple subsequent elements allowed

[3] Only taken from the configuration if not specified by the chosen Create-function

7.5 Creation of SCC Requests

CreateSECC-DiscoveryReq

Syntax	`long SCC_CreateSECCDiscoveryReq ( dword Security )`
Function	Creates a SECC Discovery Request message for sending.
Parameters	**Security:** 1 for TLS, 0 for “no transport layer security”
Returns	1 if successful, 0 else (message cannot be sent)

CreateSupported-AppProtocolReq

Syntax	`long SCC_CreateSupportedAppProtocolReq ( dword ConfigSection )`
Function	Creates a SupportedAppProtocol Request message for sending.
Parameters	**ConfigSection:** Number of the section from the test configuration file to use.
Returns	1 if successful, 0 else (message cannot be sent)

CreateSession-SetupReq

Syntax	`long SCC_CreateSessionSetupReq_Din ( byte SessionID[], byte EVCCID[] )` `long SCC_CreateSessionSetupReq_Iso ( byte SessionID[], byte EVCCID[] )`
Function	Creates a Session Setup Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **EVCCID:** 6 byte long ID of the vehicle (MAC address)
Returns	1 if successful, 0 else (message cannot be sent)

CreateService-DiscoveryReq

Syntax	`long SCC_CreateServiceDiscoveryReq_Din ( byte SessionID[] )` `long SCC_CreateServiceDiscoveryReq_Iso ( byte SessionID[] )`
Function	Creates a Service Discovery Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF.
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	ServiceScope	char[]	Scope of requested service
1	ServiceType	char[]	Type of requested service

CreateService-DetailReq	**Syntax**	`long SCC_CreateServiceDetailReq_Din ( byte SessionID[], dword ServiceId )` `long SCC_CreateServiceDetailReq_Iso ( byte SessionID[], dword ServiceId )`
	Function	Creates a Service Detail Request message for sending.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ServiceId:** ID of the requested service
	Returns	1 if successful, 0 else (message cannot be sent)

CreateService-PaymentSelection-Req	**Syntax**	`long SCC_CreateServicePaymentSelectionReq_Din ( byte SessionID[], char SelectedPaymentOption[], byte SelectedServiceIDs[], byte SelectedParameterSetIDs[], dword ServiceIDCount )` `long SCC_CreateServicePaymentSelectionReq_Iso ( byte SessionID[], char SelectedPaymentOption[], byte SelectedServiceIDs[], byte SelectedParameterSetIDs[], dword ServiceIDCount )`
	Function	Creates a Service Payment Selection / Payment Service Selection Request message for sending. > Use the short signature to load the SelectedServiceList from the XML configuration. > Use the extended signature to specify a SelectedServiceList during runtime, e.g. to match the EVSE's provided service IDs.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **SelectedPaymentOption:** "Contract" or "ExternalPayment" **SelectedServiceIDs:** IDs of the selected services. Must contain at least one entry. **SelectedParameterSetIDs:** Parameter set IDs corresponding to the SelectedServiceIDs. Array must be the same size as SelectedServiceIDs, you can use -1 to omit a parameter set ID. **ServiceIDCount:** Size of the array SelectedServiceIDs
	Returns	1 if successful, 0 else (message cannot be sent)

CreateContract-AuthenticationReq

Syntax	`long SCC_CreateContractAuthenticationReq_Din ( byte SessionID[] )` `long SCC_CreateContractAuthenticationReq_Iso ( byte SessionID[] )`
Function	Creates a Contract Authentication / Authorization Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF.
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	Id	char[]	Id of the message
1	GenChallenge	byte[]	Challenge data (16 byte)

CreatePayment-DetailsReq

Syntax	`long SCC_CreatePaymentDetailsReq_Din ( byte SessionID[], dword ConfigSection, char ContractID[] )` `long SCC_CreatePaymentDetailsReq_Iso ( byte SessionID[], dword ConfigSection, char ContractID[] )`
Function	Creates a Payment Details Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **ContractID:** ID string of the contract (eMAID)
Returns	1 if successful, 0 else (message cannot be sent)

CreateCertificate-InstallationReq

Syntax	`long SCC_CreateCertificateInstallationReq_Din ( byte SessionID[], dword ConfigSection, char DHParams[], char MessageID[] )` `long SCC_CreateCertificateInstallationReq_Iso ( byte SessionID[], dword ConfigSection, char MessageID[] )`
Function	Creates a Certificate Installation Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **DHParams:** Diffie Hellman parameter string **MessageID:** ID attribute for this message
Returns	1 if successful, 0 else (message cannot be sent)

CreateCertificate-UpdateReq

Syntax	`long SCC_CreateCertificateUpdateReq_Din ( byte SessionID[], dword ConfigSection, char ContractID[], char DHParams[], char MessageID[] )` `long SCC_CreateCertificateUpdateReq_Iso ( byte SessionID[], dword ConfigSection, char ContractID[], char MessageID[] )`
Function	Creates a Certificate Installation Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **ContractID:** ID string of the contract (eMAID) **DHParams:** Diffie Hellman parameter string **MessageID:** ID attribute for this message
Returns	1 if successful, 0 else (message cannot be sent)

CreateCharge-ParameterDiscovery-ReqAC

Syntax	`long SCC_CreateChargeParameterDiscoveryReqAC_Din ( byte SessionID[], char EnergyTransferType[], dword DepartureTime, float EAmount, float MaxVoltage, float MaxCurrent, float MinCurrent )` `long SCC_CreateChargeParameterDiscoveryReqAC_Iso ( byte SessionID[], char EnergyTransferType[], float EAmount, float MaxVoltage, float MaxCurrent, float MinCurrent )`
Function	Creates a Charge Parameter Discovery Request message for sending, using the AC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **EnergyTransferType:** Desired charging mode **EAmount:** Desired amount of energy in Wh **DepartureTime:** Intended time to finish the charging process, in “seconds from now” **MaxVoltage:** EVMaxVoltage **MaxCurrent:** EVMaxCurrent **MinCurrent:** EVMinCurrent
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	MaxEntries-SAScheduleTuple `(ISO 15118)`	long	Maximal number of entries in the SAScheduleTuple
1	DepartureTime `(ISO 15118)`	dword	See above

CreateCharge-ParameterDiscovery-ReqDC

Syntax	`long SCC_CreateChargeParameterDiscoveryReqDC_Din ( byte SessionID[], byte StatusFlags[], char ErrorCode[], char EnergyTransferType[], float MaxCurrent, float MaxVoltage)` `long SCC_CreateChargeParameterDiscoveryReqDC_Iso ( byte SessionID[], byte StatusFlags[], char ErrorCode[], char EnergyTransferType[], float MaxCurrent, float MaxVoltage)`
Function	Creates a Charge Parameter Discovery Request message for sending, using the DC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **StatusFlags:** Flags for DC_EVStatus: StatusFlags[0] = EVReady StatusFlags[1] = EVCabinConditioning StatusFlags[2] = EVRESSConditioning StatusFlags[3] = EVRESSSOC **ErrorCode:** EVErrorCode for DC_EVStatus: **EnergyTransferType:** Desired charging mode **MaxCurrent:** EVMaximumCurrentLimit **MaxVoltage:** EVMaximumVoltageLimit
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	MaxPower	float	EVMaximumPowerLimit
1	EnergyCapacity	float	Maximum supported power capacity in Wh
2	EnergyRequest	float	Amount of requested energy in Wh
3	FullSOC	long	Charge percentage to be considered as fully charged
4	BulkSOC	long	Charge percentage to be considered as the end of a fast charge process
5	MaxEntries-SAScheduleTuple (ISO 15118)	long	Maximal number of entries in the SAScheduleTuple
6	DepartureTime (ISO 15118)	dword	Intended time to finish the charging process, in “seconds from now”

CreatePower-DeliveryReqAC

Syntax	`long SCC_CreatePowerDeliveryReqAC_Din ( byte SessionID[], long ReadyToChargeState )` `long SCC_CreatePowerDeliveryReqAC_Iso ( byte SessionID[], char ChargeProgress[], long SAScheduleTupleId )`
Function	Creates a Power Delivery Request message for sending, using the AC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ReadyToChargeState:** 1 if start of charging is requested, 0 if stop of charging is requested. **ChargeProgress:** Type of action: “Start”, “Stop” or “Renegotiate” **SAScheduleTupleId:** Unique ID of a SAScheduleTuple which identifies the selected Tariff
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	ChargingProfile	complex	Desired charging profile for the current charging session (i.e. maximum amount of power drawn over time)

CreatePower-DeliveryReqDC

Syntax	`long SCC_CreatePowerDeliveryReqDC_Din ( byte SessionID[], byte StatusFlags[], char ErrorCode[], long ReadyToChargeState, long ChargingComplete )` `long SCC_CreatePowerDeliveryReqDC_Iso ( byte SessionID[], byte StatusFlags[], char ErrorCode[], char ChargeProgress[], long SAScheduleTupleId, long ChargingComplete )`
Function	Creates a Power Delivery Request message for sending, using the DC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **StatusFlags:** Flags for DC_EVStatus: StatusFlags[0] = EVReady StatusFlags[1] = EVCabinConditioning StatusFlags[2] = EVRESSConditioning StatusFlags[3] = EVRESSSOC **ErrorCode:** EVErrorCode for DC_EVStatus: **ReadyToChargeState:** 1 if start of charging is requested, 0 if stop of charging is requested. **ChargeProgress:** Type of action: “Start”, “Stop” or “Renegotiate” **SAScheduleTupleId:** Unique ID of a SAScheduleTuple which identifies the selected Tariff **ChargingComplete:** 1, if the battery is completely charged; otherwise 0.
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	ChargingProfile	complex	Desired charging profile for the current charging session (i.e. maximum amount of power drawn over time)
1	BulkCharging-Complete	long	1, if the bulk charging operation is complete; otherwise 0.

CreateCable-CheckReq

Syntax	`long SCC_CreateCableCheckReq_Din ( byte SessionID[], byte StatusFlags[], char ErrorCode[] )` `long SCC_CreateCableCheckReq_Iso ( byte SessionID[], byte StatusFlags[], char ErrorCode[] )`
Function	Creates a Cable Check Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **StatusFlags: F**lags for DC_EVStatus: StatusFlags[0] = EVReady StatusFlags[1] = EVCabinConditioning StatusFlags[2] = EVRESSConditioning StatusFlags[3] = EVRESSSOC **ErrorCode:** EVErrorCode for DC_EVStatus:
Returns	1 if successful, 0 else (message cannot be sent)

CreatePre-ChargeReq

Syntax	`long SCC_CreatePreChargeReq_Din ( byte SessionID[], byte StatusFlags[], char ErrorCode[], float TargetVoltage, float TargetCurrent )` `long SCC_CreatePreChargeReq_Iso ( byte SessionID[], byte StatusFlags[], char ErrorCode[], float TargetVoltage, float TargetCurrent )`
Function	Creates a PreCharge Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **StatusFlags:** Flags for DC_EVStatus: StatusFlags[0] = EVReady StatusFlags[1] = EVCabinConditioning StatusFlags[2] = EVRESSConditioning StatusFlags[3] = EVRESSSOC **ErrorCode:** EVErrorCode for DC_EVStatus: **TargetVoltage:** EVTargetVoltage: **TargetCurrent:** EVTargetCurrent
Returns	1 if successful, 0 else (message cannot be sent)

CreateCurrent-DemandReq

Syntax	`long SCC_CreateCurrentDemandReq_Din ( byte SessionID[], byte StatusFlags[], char ErrorCode[], float TargetVoltage, float TargetCurrent, long ChargingComplete )` `long SCC_CreateCurrentDemandReq_Iso ( byte SessionID[], byte StatusFlags[], char ErrorCode[], float TargetVoltage, float TargetCurrent, long ChargingComplete )`
Function	Creates a Current Demand Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **StatusFlags:** Flags for DC_EVStatus: StatusFlags[0] = EVReady StatusFlags[1] = EVCabinConditioning StatusFlags[2] = EVRESSConditioning StatusFlags[3] = EVRESSSOC **ErrorCode:** EVErrorCode for DC_EVStatus: **TargetVoltage:** EVTargetVoltage: **TargetCurrent:** EVTargetCurrent **ChargingComplete:** 1, if the battery is completely charged; otherwise 0.
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	MaxVoltage	float	EVMaximumVoltageLimit
1	MaxCurrent	float	EVMaximumCurrentLimit
2	MaxPower	float	EVMaximumPowerLimit
3	BulkCharging-Complete	long	1, if the bulk charging operation is complete; otherwise 0.
4	RemainingTime-ToFullSOC	float	Remaining time until full charging condition in seconds.
5	RemainingTime-ToBulkSOC	float	Remaining time until bulk charging condition in seconds.

CreateWelding-DetectionReq

Syntax	`long SCC_CreateWeldingDetectionReq_Din ( byte SessionID[], byte StatusFlags[], char ErrorCode[] )` `long SCC_CreateWeldingDetectionReq_Iso ( byte SessionID[], byte StatusFlags[], char ErrorCode[] )`
Function	Creates a Welding Detection Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **StatusFlags:** Flags for DC_EVStatus: StatusFlags[0] = EVReady StatusFlags[1] = EVCabinConditioning StatusFlags[2] = EVRESSConditioning StatusFlags[3] = EVRESSSOC **ErrorCode:** EVErrorCode for DC_EVStatus:
Returns	1 if successful, 0 else (message cannot be sent)

CreateCharging-StatusReq

Syntax	`long SCC_CreateChargingStatusReq_Din ( byte SessionID[] )` `long SCC_CreateChargingStatusReq_Iso ( byte SessionID[] )`
Function	Creates a Charging Status Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF.
Returns	1 if successful, 0 else (message cannot be sent)

CreateMetering-ReceiptReq

Syntax	`long SCC_CreateMeteringReceiptReq_Din ( byte SessionID[], long SAScheduleTupleId, char MeterID[], char ReceiptID[] )` `long SCC_CreateMeteringReceiptReq_Iso ( byte SessionID[], long SAScheduleTupleId, char MeterID[], char ReceiptID[] )`
Function	Creates a Metering Receipt Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **SAScheduleTupleId:** Unique ID of a SAScheduleTuple which identifies the selected Tariff **MeterID:** Unique ID of the meter **ReceiptID:** Unique ID of the MeteringReceiptReq message
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	MeterReading	var[1)]	Current meter reading in Wh
1	SigMeterReading	byte[]	Signature of the meter reading (length 64)
2	MeterStatus	long	Current status of the meter
3	TMeter	long	Timestamp of the current SECC time

[1)] for ISO 15118, use type long, else use type double

CreateSession-StopReq

Syntax	`long SCC_CreateSessionStopReq_Din ( byte SessionID[] )` `long SCC_CreateSessionStopReq_Iso ( byte SessionID[], char ChargingSession[] )`
Function	Creates a Session Stop Request message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ChargingSession:** Type of action, “Terminate” or “Pause”
Returns	1 if successful, 0 else (message cannot be sent)

7.6 Creation of SCC Responses

CreateSECC-DiscoveryRes

Syntax	`long SCC_CreateSECCDiscoveryRes ( char IpAddress[],dword Port, dword Security )`
Function	Creates a SECC Discovery Response message for sending. Depending on the function call, the IP Address is either interpreted as an IPv4 address or an IPv6 address.
Parameters	**IpAddress:** String representation of the EVSE’s IP address in the usual notation. **Port:** Target port number **Security:** 1 for TLS, 0 for “no transport layer security”
Returns	1 if successful, 0 else (message cannot be sent)

CreateSupported-AppProtocolRes

Syntax	`long SCC_CreateSupportedAppProtocolRes ( char ResponseCode[] )`
Function	Creates a SupportedAppProtocol Response message for sending.
Parameters	**ResponseCode:** Acknowledgement status of the message
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	SchemaId	long	Unique ID of one of the vehicle’s supported protocols

CreateSession-SetupRes

Syntax	`long SCC_CreateSessionSetupRes_Din ( byte SessionID[], char ResponseCode[], char EVSEID[] )` `long SCC_CreateSessionSetupRes_Iso ( byte SessionID[], char ResponseCode[], char EVSEID[] )`
Function	Creates a Session Setup Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **EVSEID:** Unique ID of the charge point
Returns	1 if successful, 0 else (message cannot be sent)

Caution: For DIN 70121, EVSEID is interpreted as a 64 bit number, and thus must not contain any letters.

Optional parameters

Index	Name	Type	Description
0	Timestamp	long	Timestamp of the current charge point time

CreateService-DiscoveryRes

Syntax	`long SCC_CreateServiceDiscoveryRes_Din ( byte SessionID[], dword ConfigSection, char ResponseCode[] )` `long SCC_CreateServiceDiscoveryRes_Iso ( byte SessionID[], dword ConfigSection, char ResponseCode[] )`
Function	Creates a Service Discovery Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **ResponseCode:** Acknowledgement status of the message
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	ServiceList	complex	List of offered services besides charging services

CreateService-DetailRes

Syntax	`long SCC_CreateServiceDetailRes_Din ( byte SessionID[], char ResponseCode[], dword ServiceID )` `long SCC_CreateServiceDetailRes_Iso ( byte SessionID[], char ResponseCode[], dword ServiceID )`
Function	Creates a Service Detail Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **ServiceId:** ID of the requested service
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	ServiceParameter-List	complex	List of parameters for the specific serviceID

CreateService-PaymentSelection-Res

Syntax	`long SCC_CreateServicePaymentSelectionRes_Din ( byte SessionID[], char ResponseCode[] )` `long SCC_CreateServicePaymentSelectionRes_Iso ( byte SessionID[], char ResponseCode[] )`
Function	Creates a Service Payment Selection / Payment Service Selection Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message
Returns	1 if successful, 0 else (message cannot be sent)

CreateContract-AuthenticationRes

Syntax	`long SCC_CreateContractAuthenticationRes_Din ( byte SessionID[], char ResponseCode[], long EVSEProcessing )` `long SCC_CreateContractAuthenticationRes_Iso ( byte SessionID[], char ResponseCode[], long EVSEProcessing )`
Function	Creates a Contract Authentication / Authorization Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **EVSEProcessing:** 0 if “Finished”, 1 if “Ongoing” 2 if “Ongoing_WaitingForCustomerInteraction” (ISO 15118)
Returns	1 if successful, 0 else (message cannot be sent)

CreatePayment-DetailsRes	**Syntax**	`long SCC_CreatePaymentDetailsRes_Din ( byte SessionID[], char ResponseCode[], byte GenChallenge[], long Timestamp )` `long SCC_CreatePaymentDetailsRes_Iso ( byte SessionID[], char ResponseCode[], byte GenChallenge[], long Timestamp )`
	Function	Creates a Payment Details Response message for sending.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **GenChallenge:** Challenge data (max. 16 byte) **Timestamp:** Timestamp of the current charge point time
	Returns	1 if successful, 0 else (message cannot be sent)

CreateCertificate-InstallationRes	**Syntax**	`long SCC_CreateCertificateInstallationRes_Din ( byte SessionID[], dword ConfigSection, char ResponseCode[], char EncryptedPrivateKey[], char DHParams[], char ContractID[], char MessageID[] )` `long SCC_CreateCertificateInstallationRes_Iso ( byte SessionID[], dword ConfigSection, char ResponseCode[], byte EncryptedPrivateKey[], byte DHParams[], char ContractID[] )`
	Function	Creates a Certificate Installation Response message for sending.
	Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **ResponseCode:** Acknowledgement status of the message **EncryptedPrivateKey:** The private key that belongs to the new certificate for signature purposes **DHParams:** Diffie Hellman parameter string **ContractID:** ID string of the contract (eMAID) **MessageId:** ID attribute for this message
	Returns	1 if successful, 0 else (message cannot be sent)

CreateCertificate-UpdateRes

Syntax	`long SCC_CreateCertificateUpdateRes_Din ( byte SessionID[], dword ConfigSection, char ResponseCode[], char EncryptedPrivateKey[], char DHParams[], char ContractID[], long RetryCounter, char MessageID[] )` `long SCC_CreateCertificateUpdateRes_Iso ( byte SessionID[], dword ConfigSection, char ResponseCode[],byte EncryptedPrivateKey[], byte DHParams[], char ContractID[] )`
Function	Creates a Certificate Installation Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **ResponseCode:** Acknowledgement status of the message **EncryptedPrivateKey:** The private key that belongs to the new certificate for signature purposes **DHParams:** Diffie Hellman parameter string **ContractID:** ID string of the contract (eMAID) **RetryCounter:** In case of failure, this denotes when the EVCC should try to get the new Certificate again (number of days) **MessageId:** ID attribute for this message
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	RetryCounter (ISO 15118)	long	See above

CreateCharge-ParameterDiscovery-ResAC

Syntax	`long SCC_CreateChargeParameterDiscoveryResAC_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[], long EVSEProcessing, float CurrentAndVoltageLimits[])` `long SCC_CreateChargeParameterDiscoveryResAC_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[], long EVSEProcessing, float NominalVoltage, float MaxCurrent )`
Function	Creates a Charge Parameter Discovery Response message for sending, using the AC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ConfigSection:** Number of the section from the test configuration file to use. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for AC_EVSEStatus) **StatusFlags:** Flags for AC_EVSEStatus: StatusFlags[0] = PowerSwitchClosed StatusFlags[1] = RCD **Notification:** Notification about an action that the charge point wants the vehicle to perform (for AC_EVSEStatus) **EVSEProcessing:** 0 if “Finished”, 1 if “Ongoing” 2 if “Ongoing_WaitingForCustomerInteraction” (ISO 15118) **CurrentAndVoltageLimits:** Electrical limit values CurrentAndVoltageLimits[0] = MaxVoltage CurrentAndVoltageLimits[1] = MaxCurrent CurrentAndVoltageLimits[2] = MinCurrent **NominalVoltage:** Line voltage supported by the EVSE **MaxCurrent:** EVSEMaxCurrent
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	SAScheduleList	long	Schedule / tariff list from secondary actor

CreateCharge-ParameterDiscovery-ResDC

Syntax	`long SCC_CreateChargeParameterDiscoveryResDC_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], long EVSEProcessing, float CurrentAndVoltageLimits[], float PeakCurrentRipple)` `long SCC_CreateChargeParameterDiscoveryResDC_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], long EVSEProcessing, float CurrentAndVoltageLimits[], float PeakCurrentRipple)`
Function	Creates a Charge Parameter Discovery Response message for sending, using the DC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for DC_EVSEStatus) **StatusCode:** Internal state of the EVSE (for DC_EVSEStatus) **Notification:** Notification about an action that the charge point wants the vehicle to perform (for DC_EVSEStatus) **EVSEProcessing:** 0 if “Finished”, 1 if “Ongoing” 2 if “Ongoing_WaitingForCustomerInteraction” (ISO 15118) **CurrentAndVoltageLimits:** Electrical limit values CurrentAndVoltageLimits[0] = EVSEMaximumCurrentLimit CurrentAndVoltageLimits[1] = EVSEMaximumVoltageLimit CurrentAndVoltageLimits[2] = EVSEMinimumCurrentLimit CurrentAndVoltageLimits[3] = EVSEMinimumVoltageLimit Only for ISO 15118: CurrentAndVoltageLimits[4] = EVSEMaximumPowerLimit **PeakCurrentRipple:** Peak-to-peak magnitude of the current ripple
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	MaxPower	float	EVSEMaximumPowerLimit
1	CurrentRegulation-Tolerance	float	Absolute magnitude of the regulation tolerance
2	EnergyToBeDelivered	float	Amount of energy to be delivered in Wh
3	SAScheduleList	long	Schedule / tariff list from secondary actor
4	EVSEIsolationStatus	char[]	Indicates the isolation condition

CreatePower-DeliveryResAC

Syntax	`long SCC_CreatePowerDeliveryResAC_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[] )` `long SCC_CreatePowerDeliveryResAC_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[] )`
Function	Creates a Power Delivery Response message for sending, using the AC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for AC_EVSEStatus) **StatusFlags:** Flags for AC_EVSEStatus: StatusFlags[0] = PowerSwitchClosed StatusFlags[1] = RCD **Notification:** Notification about an action that the charge point wants the vehicle to perform (for AC_EVSEStatus)
Returns	1 if successful, 0 else (message cannot be sent)

CreatePower-DeliveryResDC

Syntax	`long SCC_CreatePowerDeliveryResDC_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[] )` `long SCC_CreatePowerDeliveryResDC_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[] )`
Function	Creates a Power Delivery Response message for sending, using the DC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for DC_EVSEStatus) **StatusCode:** Internal state of the EVSE (for DC_EVSEStatus) **Notification:** Notification about an action that the charge point wants the vehicle to perform (for DC_EVSEStatus)
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	IsolationStatus	char[]	Current isolation condition

CreateCable-CheckRes

Syntax	`long SCC_CreateCableCheckRes_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], long EVSEProcessing )` `long SCC_CreateCableCheckRes_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], long EVSEProcessing )`
Function	Creates a Cable Check Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for DC_EVSEStatus) **StatusCode:** Internal state of the EVSE (for DC_EVSEStatus) **Notification:** Notification about an action that the charge point wants the vehicle to perform (for DC_EVSEStatus) **EVSEProcessing:** 0 if “Finished”, 1 if “Ongoing” 2 if “Ongoing_WaitingForCustomerInteraction” (ISO 15118)
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	IsolationStatus	char[]	Current isolation condition

CreatePre-ChargeRes

Syntax	`long SCC_CreatePreChargeRes_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], float PresentVoltage )` `long SCC_CreatePreChargeRes_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], float PresentVoltage )`
Function	Creates a PreCharge Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for DC_EVSEStatus) **StatusCode**: Internal state of the EVSE (for DC_EVSEStatus) **Notification:** Notification about an action that the charge point wants the vehicle to perform (for DC_EVSEStatus) **PresentVoltage:** EVSEPresentVoltage
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	IsolationStatus	char[]	Current isolation condition

CreateCurrent-DemandRes

Syntax	`long SCC_CreateCurrentDemandRes_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], float PresentVoltage, float PresentCurrent, byte LimitAchievedFlags[] )` `long SCC_CreateCurrentDemandRes_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], float PresentValues[], byte LimitAchievedFlags[], char EVSEID[], long SAScheduleTupleID )`
Function	Creates a Current Demand Reponse message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for DC_EVSEStatus) **StatusCode:** Internal state of the EVSE (for DC_EVSEStatus) **Notification:** Notification about an action that the charge point wants the vehicle to perform (for DC_EVSEStatus) **PresentVoltage:** EVSEPresentVoltage **PresentCurrent:** EVSEPresentCurrent **PresentValues:** Replaces the two values above: PresentValues [0] = EVSEPresentVoltage PresentValues [1] = EVSEPresentCurrent **LimitAchievedFlags:** Flags that indicate if any limit is achieved: LimitAchievedFlags [0] = EVSECurrentLimitAchieved LimitAchievedFlags [1] = EVSEVoltageLimitAchieved LimitAchievedFlags [2] = EVSEPowerLimitAchieved **EVSEID:** Unique ID of the charge point **SAScheduleTupleID:** ID of the selected SAScheduleTuple
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	IsolationStatus	char[]	Current isolation condition
1	MaxVoltage	float	EVSEMaximumVoltageLimit
2	MaxCurrent	float	EVSEMaximumCurrentLimit
3	MaxPower	float	EVSEMaximumPowerLimit
`ISO 15118` only parameters:			
4	ReceiptRequired	long	Indicates if the vehicle is required to sent a MeteringReceiptReq
5	MeterID	char[]	Unique ID of the meter
6	MeterReading	var[1)]	Current meter reading in Wh
7	SigMeterReading	byte[]	Signature of the meter reading (length 64)
8	MeterStatus	long	Current status of the meter
9	TMeter	long	Timestamp of the current SECC time

[1)] for ISO 15118, use type long, else use type double

Caution: Due to the requirements of the schema, the element “MeterID” is mandatory if any of the optional meter-related parameters are to be supplied.

CreateWelding-DetectionRes

Syntax	`long SCC_CreateWeldingDetectionRes_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], float PresentVoltage )` `long SCC_CreateWeldingDetectionRes_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[], float PresentVoltage )`
Function	Creates a Welding Detection Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for DC_EVSEStatus) **StatusCode:** Internal state of the EVSE (for DC_EVSEStatus) **Notification:** Notification about an action that the charge point wants the vehicle to perform (for DC_EVSEStatus) **PresentVoltage:** EVSEPresentVoltage
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	IsolationStatus	char[]	Current isolation condition

CreateCharging-StatusRes

Syntax	`long SCC_CreateChargingStatusRes_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[], char EVSEID[], long SAScheduleTupleId, long ReceiptRequired )` `long SCC_CreateChargingStatusRes_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[], char EVSEID[], long SAScheduleTupleId )`
Function	Creates a Charging Status Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for AC_EVSEStatus) **StatusFlags:** Flags for AC_EVSEStatus: StatusFlags[0] = PowerSwitchClosed StatusFlags[1] = RCD **Notification:** Notification about an action that the charge point wants the vehicle to perform (for AC_EVSEStatus) **EVSEID:** Unique ID of the charge point **SAScheduleTupleId:** Unique ID of a SAScheduleTuple which identifies the selected Tariff **ReceiptRequired:** 1 if a MeteringReceiptReq message is expected next, else 0
Returns	1 if successful, 0 else (message cannot be sent)

Caution: EVSEID is interpreted as a 64 bit number, and thus must not contain any letters.

Optional parameters

Index	Name	Type	Description
0	MeterID	char[]	Unique ID of the meter
1	MeterReading	var[1)]	Current meter reading in Wh
2	SigMeterReading	byte[]	Signature of the meter reading (length 64)
3	MeterStatus	long	Current status of the meter
4	TMeter	long	Timestamp of the current SECC time
5	MaxCurrent	float	EVSEMaxCurrent
6	ReceiptRequired `(ISO 15118)`	long	Indicates if the vehicle is required to sent a MeteringReceiptReq

[1)] for ISO 15118, use type long, else use type double

Caution: Due to the requirements of the schema, the element “MeterID” is mandatory if any of the optional meter-related parameters are to be supplied.

CreateMetering-ReceiptResAC

Syntax	`long SCC_CreateMeteringReceiptResAC_Din ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[] )` `long SCC_CreateMeteringReceiptResAC_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, byte StatusFlags[], char Notification[] )`
Function	Creates a Metering Receipt Response message for sending, using the AC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for AC_EVSEStatus) **StatusFlags:** Flags for AC_EVSEStatus: StatusFlags[0] = PowerSwitchClosed StatusFlags[1] = RCD **Notification:** Notification about an action that the charge point wants the vehicle to perform (for AC_EVSEStatus)
Returns	1 if successful, 0 else (message cannot be sent)

CreateMetering-ReceiptResDC

Syntax (ISO 15118)	`long SCC_CreateMeteringReceiptResDC_Iso ( byte SessionID[], char ResponseCode[], long NotificationMaxDelay, char StatusCode[], char Notification[] )`
Function	Creates a Metering Receipt Response message for sending, using the DC syntax.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message **NotificationMaxDelay:** Time until the vehicle is expected to react on the notification (for DC_EVSEStatus) **StatusCode**: Internal state of the EVSE (for DC_EVSEStatus) **Notification:** Notification about an action that the charge point wants the vehicle to perform (for DC_EVSEStatus)
Returns	1 if successful, 0 else (message cannot be sent)

Optional parameters

Index	Name	Type	Description
0	IsolationStatus	char[]	Current isolation condition

CreateSession-StopRes

Syntax	`long SCC_CreateSessionStopRes_Din ( byte SessionID[], char ResponseCode[] )` `long SCC_CreateSessionStopRes_Iso ( byte SessionID[], char ResponseCode[] )`
Function	Creates a Session Stop Response message for sending.
Parameters	**SessionID:** 8-byte long SessionID of SCC connection, range: 0 - 0xFF FF FF FF FF FF FF FF. **ResponseCode:** Acknowledgement status of the message
Returns	1 if successful, 0 else (message cannot be sent)

7.7 Optional Parameters and Sending

The following functions are needed for modification and sending of the message created using the functions of section 7.5 and 7.6.

SetOptional-Parameter

Syntax	`long SCC_SetOptionalParameter ( dword Parameter, long ParameterValue )`
	`long SCC_SetOptionalParameter ( dword Parameter, float ParameterValue )`
	`long SCC_SetOptionalParameter ( dword Parameter, char ParameterValue[] )`
	`long SCC_SetOptionalParameter ( dword Parameter, byte ParameterValue[] )`

Function	Sets an optional parameter after creating a message.
Parameters	**Parameter:** ID number of the parameter, as detoned in the lists of optional parameters below each Create-function **ParameterValue:** Desired value of the parameter in the matching data type
Returns	1 if successful, 0 else

Note: Please make sure to use the correct value type for the respective parameter; e.g. an integer will not be automatically converted to a string if the parameter type is char[]. For float parameters, use float notation (e.g. “0.0”, “12.8”). The parameter types are detoned in the lists of optional parameters below each Create-function.

SetOptional-ParameterUnsigned

Syntax	`long SCC_SetOptionalParameterUnsigned ( dword Parameter, dword ParameterValue )`
Function	Sets an optional parameter using a DWORD value after creating a message.
Parameters	**Parameter:** ID number of the parameter, as detoned in the lists of optional parameters below each Create-function **ParameterValue:** Desired value of the parameter in the matching data type
Returns	1 if successful, 0 else

Note: Parameters marked as “long” or “dword” can all be access both with this function and with `SCC_SetOptionalParameter(dword, long)`. Usage of this function is only mandatory if the whole 32 bit unsigned number range is needed.

SetOptional-Parameter-FromConfig

Syntax	`long SCC_SetOptionalParameterFromConfig ( dword Parameter, dword ConfigSection )`
Function	Sets an optional parameter of type “complex” by referring to one of the “TestConfiguration” sections.
Parameters	**Parameter:** ID number of the parameter, as detoned in the lists of optional parameters below each Create-function **ConfigSection:** ID of the config section where the parameter can be found
Returns	1 if successful, 0 else

SetPreparedMsg-HeaderData

Syntax	`void SCC_SetPreparedMsgHeaderData ( byte Header[] )`
Function	Overwrites the SDP/V2G header for a prepared message. (Refer to the DIN / ISO specifications for the header layout.)
Parameters	**Header:** 8 byte V2G header
Returns	-

SetPreparedMsg-FaultNotification

Syntax	`long SCC_SetPreparedMsgFaultNotification ( char FaultCode[], char FaultMessage[] )`
Function	Sets the fault code and fault message for a prepared message
Parameters	**FaultCode:** Desired fault code, which must be a valid enum value according to the specification **FaultMsg:** Desired fault message string. If this string is empty, the optional message element is omitted.
Returns	1 if successful, 0 else

SendPrepared-Message

Syntax	`long SCC_SendPreparedMessage ( )` `long SCC_SendPreparedMessage ( dword ConfigSection )`
Function	Sends a message created using one of the Create-functions. Use the function call **with** parameter “ConfigSection” to create a **signed** message using the certificate name from the XML config. Else an **unsigned** message is sent.
Parameters	**ConfigSection:** ID of the config section where the certificates can be found
Returns	1 if successful, 0 else

Note: If multiple Create-functions are called without using `SCC_SendPreparedMessage()` in between, all prior messages are discarded. Only the last created message is available for sending.

Note: Only messages that shall be signed according to the ISO 15118 specification may be sent as signed message. For all other messages, there is no difference between the two function calls.

GetPrepared-EXIMessage

Syntax	`long SCC_GetPreparedEXIMessage ( byte data[], dword& dataLength )` `long SCC_GetPreparedEXIMessage ( byte data[], dword& dataLength, dword ConfigSection )`
Function	Finalizes a message without sending. The message data is returned to an output buffer instead. Usually this data consists of the V2G header and the EXI encoded V2G payload.
Parameters	**Data:** Output buffer for the message. Make sure to use a buffer that is large enough (some messages may take up to 10 kB). **DataLength:** Length of the copied data **ConfigSection:** ID of the config section where the certificates can be found
Returns	1 if successful, 0 else

Note: Use this function to access the EXI encoder to create reference messages, or to create V2G messages with a custom TCP header. The latter can then be sent using the features of CANoe .Ethernet.

Note: It is possible to call `SendPreparedMessage()` subsequently without preparing the message anew.

8 Appendix A: Copyright

8.1 QCA_Open_PLC_Utils_License_2013-10-15

Please note the following conditions and the disclaimer regarding the OpenPLC Utils (QCAConfigurator) of the Smart Charging Communication Package.

Nevertheless, regarding your contractual relationship towards Vector Informatik GmbH (Vector), your contract with Vector applies and remains unaffected, especially regarding Vector's Liability:

9 Appendix B: Enum Reference

9.1 Message ID

9.1.1 V2G Vehicle Requests

Message	Message ID
Legacy Request	0
SupportedAppProtocolReq	1
SessionSetupReq	2
ServiceDiscoveryReq	3
ServiceDetailReq	4
PaymentServiceSelectionReq / ServicePaymentSelectionReq	5
PaymentDetailsReq	6
AuthorizationReq / ContractAuthenticationReq	7
ChargeParameterDiscoveryReq	8
PowerDeliveryReq	9
ChargingStatusReq	10
MeteringReceiptReq	11
CableCheckReq	12
PreChargeReq	13
CurrentDemandReq	14
WeldingDetectionReq	15
SessionStopReq	16
CertificateUpdateReq	17
CertificateInstallationReq	18

9.1.2 V2G Charge Point Responses

Message	Message ID
Legacy Response	0
SupportedAppProtocolRes	1
SessionSetupRes	2
ServiceDiscoveryRes	3
ServiceDetailRes	4
PaymentServiceSelectionRes / ServicePaymentSelectionRes	5
PaymentDetailsRes	6
AuthorizationRes / ContractAuthenticationRes	7
ChargeParameterDiscoveryRes	8
PowerDeliveryRes	9
ChargingStatusRes	10

Message	Message ID
MeteringReceiptRes	11
CableCheckRes	12
PreChargeRes	13
CurrentDemandRes	14
WeldingDetectionRes	15
SessionStopRes	16
CertificateUpdateRes	17
CertificateInstallationRes	18

9.1.3 Other messages

In cases where a Message ID is provided for a non-V2G message, the following applies:

SLAC

For SLAC messages (MME frames), the message ID provided is the frame's MMType, e.g. 0x6064 for CM_SLAC_Parm_Req.

SECC Discovery

For SECC Discovery messages, the message ID is the V2GTP payload type, i.e.

0x9000 for SECC Discovery Request
0x9001 for SECC Discovery Response

9.1.4 Combined message ID

Note: The combined message ID is used by SCC Observer and SCC Monitor for the variable "LastMessageType", while in CAPL functions the specific IDs above are used.

Message	Message ID
SECCDiscoveryReq/-Res	1/2
SupportedAppProtocolReq/-Res	3/4
SessionSetupReq/-Res	5/6
ServiceDiscoveryReq/-Res	7/8
ServiceDetailReq/-Res	9/10
PaymentServiceSelectionReq/-Res ServicePaymentSelectionReq/-Res	11/12
PaymentDetailsReq/-Res	13/14
AuthorizationReq/-Res ContractAuthenticationReq/-Res	15/16
ChargeParameterDiscoveryReq/-Res	17/18
(unused)	19/20
PowerDeliveryReq/-Res	21/22
(unused)	23/24
MeteringReceiptReq/-Res	25/26
CableCheckReq/-Res	27/28
PreChargeReq/-Res	29/30

Message	Message ID
CurrentDemandReq/-Res	31/32
WeldingDetectionReq/-Res	33/34
SessionStopReq/-Res	37/38
ChargingStatusReq/-Res	39/40
CertificateUpdateReq/-Res	41/42
CertificateInstallationReq/-Res	43/44
CM_SLAC_Parm.Req/.Cnf	45/46
CM_Start_Atten_Char.Ind	47
CM_MNBC_Sound.Ind	48
CM_Atten_Char.Ind/.Rsp	49/50
CM_Validate.Req/.Cnf	51/52
CM_SLAC_Match.Req/.Cnf	53/54
VS_Atten_Char.Ind (deprecated)	55
VS_Set_Key.Req/.Cnf (deprecated)	56/57
CM_Atten_Profile.Ind	58
CM_Set_Key.Req/.Cnf	59/60
VS_Pl_LnkStatus.Req/.Cnf	61/62
CM_Amp_Map.Req/.Cnf	63/64

9.2 State ID

9.2.1 Vehicle States - General

Description / corresponding messages	State ID
Initial	0
Error	200

9.2.2 Vehicle States - Message related

Description / corresponding messages	State ID
Session Setup	1
Service Discovery	2
Service Details	3
Service / Payment Selection	4
Payment Details	5
Authentication / Authorization	6
Charge Parameter Discovery	7
Power Delivery	8
Charging Status	9
Metering Receipt	10
Cable Check	11

Description / corresponding messages	State ID
PreCharge	12
Current Demand	13
Welding Detection	14
Session Stop	15
Certificate Update	16

9.2.3 Charge Point States - General

Description / corresponding messages	State ID
Initial	0
Error	200
Wait (end of protocol)	201

9.2.4 Charge Point States – Message related

Description / corresponding messages	State ID
Session Setup	1
Service Discovery	2
Service / Payment Selection	3
Payment Details	4
Payment Details / Certificate transfer	5
Authentication / Authorization	6
Charge Parameter Discovery	7
Power Delivery	8
Charging Status	9
Metering Receipt	10
Cable Check	11
PreCharge	12
Current Demand	13
Welding Detection	14
Welding Detection or Renegotiation	15
Renegotiation	16
Session Stop	17
Session Stop or Renegotiation	18

10 Index

E

G

T

V

W

More Information

- News
- Products
- Demo Software
- Support
- Training Classes
- Addresses

www.vector.com