Document Title	Specification of VDP Communication Module Remote
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	1139

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R25-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2025-11-27	R25-11	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

Known Limitations

There are no limitations.

1 Introduction and functional overview

This specification describes the functionality, the API and the configuration for the AUTOSAR Basic Software module VDP-CM Remote[1].

The VDP-CM Remote implements the remote ECU side of the Vehicle Data Protocol (VDP) as specified in [1, VDP PRS].

[1] CM stands for Communication Module, see chapter 2.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the VDP-CM Remote module that are not included in the [2, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
CM	Communication Module
DCA	Data Class Adapter
DP	Data point
VDP	Vehicle Data Protocol
VdpCmR	Technical name of the module *VDP-CM Remote*

Table 2.1: Acronyms and abbreviations used in the scope of this Document

2.1 Definition of terms

Terms:	Description:
Data point	A data point refers to a data resource inside the vehicle with a well-defined semantic that can be collected (e.g. vehicle speed).
Data point sample	A data point sample refers to the value of a data point acquired at a specific point in time (e.g. vehicle speed at 12 am).
DCA Remote	A DCA Remote refers to a Data Class Adapter Remote. The Data Class Adapter Remotes are directly connected to a CM Remote and can directly access data points on the other side. They can be realized e.g. by integrator code, AUTOSAR Complex Device Drivers or a specific SWC.
VDP Slot ID	The VDP Slot ID refers to a dynamically associated ID to an abstract data point during the lifetime of a data collection task. The dynamic association is part of the protocol VDP and is established using the add dynamic configuration command.

Table 2.2: Definition of terms in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Vehicle Data Protocol Specification
AUTOSAR_FO_PRS_VDP

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[4] Requirements on Vehicle Data Protocol
AUTOSAR_FO_RS_VDP

[5] Specification of PDU Router
AUTOSAR_CP_SWS_PDURouter

[6] Specification of Synchronized Time-Base Manager
AUTOSAR_CP_SWS_SynchronizedTimeBaseManager

[7] Specification of NVRAM Manager
AUTOSAR_CP_SWS_NVRAMManager

[8] Specification of Default Error Tracer
AUTOSAR_CP_SWS_DefaultErrorTracer

[9] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3, SWS BSW General], which is also valid for VDP-CM Remote.

Furthermore, since the VDP-CM Remote implements VDP, the specifications [1, VDP PRS] and [4, VDP RS] are valid for the VDP-CM Remote.

Thus, the previously mentioned specifications shall be considered as additional and required specification for VDP-CM Remote.

4 Constraints and assumptions

4.1 Limitations

None

4.2 Applicability to car domains

See introduction 1

5 Dependencies to other modules

5.1 PduR

In order to receive and transmit VDP messages on the communication bus, the VDP-CM Remote module interacts with the PDU Router [5].

5.2 StbM

In order to get a synchronized time value (Local Time Base derived from Global Time Base), the StbM module [6] is used.

5.3 NvM

In order to load and store dynamic configurations, the NvM module [7] can optionally be used.

5.4 Det

In order to be able to report development errors and to forward this information through DET to the communication bus, the VDP-CM Remote module interacts with the DET module [8]. However, the interaction with DET is optional.

6 Requirements Tracing

The following tables reference the requirements specified in [4, RS VDP] and [9, RS BSWGeneral] and links to the fulfillment of these.

Requirement	Description	Satisfied by
[FO_RS_VDP_00001]	Multi-Version Support	[CP_SWS_VdpCmR_00107] [CP_SWS_VdpCmR_00108]
[FO_RS_VDP_00002]	VDP Dynamic Configuration of Data Points	[CP_SWS_VdpCmR_00107] [CP_SWS_VdpCmR_00109] [CP_SWS_VdpCmR_00110] [CP_SWS_VdpCmR_00111] [CP_SWS_VdpCmR_00112] [CP_SWS_VdpCmR_00113] [CP_SWS_VdpCmR_00114] [CP_SWS_VdpCmR_00115] [CP_SWS_VdpCmR_00116] [CP_SWS_VdpCmR_00120] [CP_SWS_VdpCmR_00121] [CP_SWS_VdpCmR_00122] [CP_SWS_VdpCmR_00123] [CP_SWS_VdpCmR_00124] [CP_SWS_VdpCmR_00130] [CP_SWS_VdpCmR_00131] [CP_SWS_VdpCmR_00132] [CP_SWS_VdpCmR_00133] [CP_SWS_VdpCmR_00134] [CP_SWS_VdpCmR_00135] [CP_SWS_VdpCmR_00140] [CP_SWS_VdpCmR_00141] [CP_SWS_VdpCmR_00142] [CP_SWS_VdpCmR_00143] [CP_SWS_VdpCmR_00144] [CP_SWS_VdpCmR_00210] [CP_SWS_VdpCmR_00211] [CP_SWS_VdpCmR_00212] [CP_SWS_VdpCmR_00601] [CP_SWS_VdpCmR_00602] [CP_SWS_VdpCmR_00603] [CP_SWS_VdpCmR_00604] [CP_SWS_VdpCmR_00605] [CP_SWS_VdpCmR_00606] [CP_SWS_VdpCmR_00607] [CP_SWS_VdpCmR_00608] [CP_SWS_VdpCmR_00609] [CP_SWS_VdpCmR_00610] [CP_SWS_VdpCmR_00611] [CP_SWS_VdpCmR_01104] [CP_SWS_VdpCmR_01203] [CP_SWS_VdpCmR_01300] [CP_SWS_VdpCmR_01301] [CP_SWS_VdpCmR_01302] [CP_SWS_VdpCmR_01600] [CP_SWS_VdpCmR_01601] [CP_SWS_VdpCmR_01602] [CP_SWS_VdpCmR_01603] [CP_SWS_VdpCmR_01604]

△

Requirement	Description	Satisfied by
[FO_RS_VDP_00003]	VDP Different Modes for Data Sampling	[CP_SWS_VdpCmR_00109] [CP_SWS_VdpCmR_00110] [CP_SWS_VdpCmR_00111] [CP_SWS_VdpCmR_00112] [CP_SWS_VdpCmR_00113] [CP_SWS_VdpCmR_00115] [CP_SWS_VdpCmR_00120] [CP_SWS_VdpCmR_00121] [CP_SWS_VdpCmR_00122] [CP_SWS_VdpCmR_00123] [CP_SWS_VdpCmR_00124] [CP_SWS_VdpCmR_00130] [CP_SWS_VdpCmR_00131] [CP_SWS_VdpCmR_00132] [CP_SWS_VdpCmR_00140] [CP_SWS_VdpCmR_00141] [CP_SWS_VdpCmR_00142] [CP_SWS_VdpCmR_00143] [CP_SWS_VdpCmR_00144] [CP_SWS_VdpCmR_01102] [CP_SWS_VdpCmR_01103] [CP_SWS_VdpCmR_01601] [CP_SWS_VdpCmR_01602]
[FO_RS_VDP_00004]	VDP Different Modes for Data Transmission	[CP_SWS_VdpCmR_00109] [CP_SWS_VdpCmR_00110] [CP_SWS_VdpCmR_00111] [CP_SWS_VdpCmR_00112] [CP_SWS_VdpCmR_00114] [CP_SWS_VdpCmR_00115] [CP_SWS_VdpCmR_00120] [CP_SWS_VdpCmR_00121] [CP_SWS_VdpCmR_00122] [CP_SWS_VdpCmR_00123] [CP_SWS_VdpCmR_00124] [CP_SWS_VdpCmR_00131] [CP_SWS_VdpCmR_00132] [CP_SWS_VdpCmR_00140] [CP_SWS_VdpCmR_00141] [CP_SWS_VdpCmR_00142] [CP_SWS_VdpCmR_00143] [CP_SWS_VdpCmR_00144] [CP_SWS_VdpCmR_00401] [CP_SWS_VdpCmR_00403] [CP_SWS_VdpCmR_00404] [CP_SWS_VdpCmR_00405] [CP_SWS_VdpCmR_00406] [CP_SWS_VdpCmR_00407] [CP_SWS_VdpCmR_00410] [CP_SWS_VdpCmR_00701] [CP_SWS_VdpCmR_00702] [CP_SWS_VdpCmR_00705] [CP_SWS_VdpCmR_00706] [CP_SWS_VdpCmR_00707] [CP_SWS_VdpCmR_01303] [CP_SWS_VdpCmR_01304]

▽

△

Requirement	Description	Satisfied by
[FO_RS_VDP_00005]	Detection of Missing Data on the Central ECU	[CP_SWS_VdpCmR_00101] [CP_SWS_VdpCmR_00102] [CP_SWS_VdpCmR_00103] [CP_SWS_VdpCmR_00104] [CP_SWS_VdpCmR_00105] [CP_SWS_VdpCmR_00106] [CP_SWS_VdpCmR_00142] [CP_SWS_VdpCmR_00143] [CP_SWS_VdpCmR_00150] [CP_SWS_VdpCmR_00151] [CP_SWS_VdpCmR_00152] [CP_SWS_VdpCmR_00153] [CP_SWS_VdpCmR_00154] [CP_SWS_VdpCmR_00155]
[FO_RS_VDP_00006]	VDP Temporary Interruption of Data Sampling	[CP_SWS_VdpCmR_00101] [CP_SWS_VdpCmR_00102] [CP_SWS_VdpCmR_00103] [CP_SWS_VdpCmR_00104] [CP_SWS_VdpCmR_00105] [CP_SWS_VdpCmR_00106] [CP_SWS_VdpCmR_00109] [CP_SWS_VdpCmR_00115] [CP_SWS_VdpCmR_00130] [CP_SWS_VdpCmR_00131] [CP_SWS_VdpCmR_00132] [CP_SWS_VdpCmR_00150] [CP_SWS_VdpCmR_00151] [CP_SWS_VdpCmR_00152] [CP_SWS_VdpCmR_00153] [CP_SWS_VdpCmR_00154] [CP_SWS_VdpCmR_00155]
[FO_RS_VDP_00009]	VDP Efficient Representation of Meta Data	[CP_SWS_VdpCmR_00119] [CP_SWS_VdpCmR_00401] [CP_SWS_VdpCmR_00403] [CP_SWS_VdpCmR_00404] [CP_SWS_VdpCmR_00405] [CP_SWS_VdpCmR_00406] [CP_SWS_VdpCmR_00407] [CP_SWS_VdpCmR_00408] [CP_SWS_VdpCmR_00410] [CP_SWS_VdpCmR_00501] [CP_SWS_VdpCmR_00506] [CP_SWS_VdpCmR_00703] [CP_SWS_VdpCmR_00704] [CP_SWS_VdpCmR_01204]
[FO_RS_VDP_00010]	VDP Error Handling	[CP_SWS_VdpCmR_00115] [CP_SWS_VdpCmR_00116] [CP_SWS_VdpCmR_00119] [CP_SWS_VdpCmR_00213] [CP_SWS_VdpCmR_00402] [CP_SWS_VdpCmR_00407] [CP_SWS_VdpCmR_00408] [CP_SWS_VdpCmR_00409] [CP_SWS_VdpCmR_00501] [CP_SWS_VdpCmR_00503] [CP_SWS_VdpCmR_00504] [CP_SWS_VdpCmR_00505] [CP_SWS_VdpCmR_01204] [CP_SWS_VdpCmR_01303] [CP_SWS_VdpCmR_01304]

▽

△

Requirement	Description	Satisfied by
[FO_RS_VDP_00011]	VDP Control Sequence Enforcement	[CP_SWS_VdpCmR_00110] [CP_SWS_VdpCmR_00111] [CP_SWS_VdpCmR_00120] [CP_SWS_VdpCmR_00133] [CP_SWS_VdpCmR_00134] [CP_SWS_VdpCmR_00135] [CP_SWS_VdpCmR_00143] [CP_SWS_VdpCmR_00144]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[CP_SWS_VdpCmR_00100] [CP_SWS_VdpCmR_01101] [CP_SWS_VdpCmR_01200]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[CP_SWS_VdpCmR_02000]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[CP_SWS_VdpCmR_01100]
[SRS_BSW_00337]	Classification of development errors	[CP_SWS_VdpCmR_01000] [CP_SWS_VdpCmR_01001] [CP_SWS_VdpCmR_01002] [CP_SWS_VdpCmR_01003]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[CP_SWS_VdpCmR_01101] [CP_SWS_VdpCmR_01200]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[CP_SWS_VdpCmR_01101] [CP_SWS_VdpCmR_01200]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[CP_SWS_VdpCmR_01001] [CP_SWS_VdpCmR_01002] [CP_SWS_VdpCmR_01003]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[CP_SWS_VdpCmR_01400]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[CP_SWS_VdpCmR_01500] [CP_SWS_VdpCmR_01501]
[SRS_BSW_00402]	Each module shall provide version information	[CP_SWS_VdpCmR_01202]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[CP_SWS_VdpCmR_01101] [CP_SWS_VdpCmR_01200]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[CP_SWS_VdpCmR_01201]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 Initialization

[CP_SWS_VdpCmR_00100] Initialization

Upstream requirements: SRS_BSW_00101

⌈A call to `VdpCmR_Init` initializes all internal variables and sets the `VdpCmR` module to the initialized state.⌋

7.2 Communication with PduR

[CP_SWS_VdpCmR_00150] Communication configuration

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The `VdpCmR` shall use the content of *VdpCmRCsl* to configure its communication and security layer.⌋

[CP_SWS_VdpCmR_00151] Unsecure and secure PDU reception

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The `VdpCmR` shall receive PDUs that are transmitted unsecured (mandatory) as well as secured (optional, defined project specific).⌋

[CP_SWS_VdpCmR_00152] Rx PDU configuration

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The `VdpCmR` shall use the general purpose PDU defined by container *VdpCmRCslRxPdu* for reception from the PduR.⌋

In case of secured transmission are the PDUs already processed by SecOC and only the unsecured PDU part is passed from the PduR to `VdpCmR`. So the `VdpCmR` does not see any difference between secured and unsecured PDUs at reception.

[CP_SWS_VdpCmR_00153] Unsecure and secure PDU transmission

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The `VdpCmR` shall be able to transmit PDUs unsecured (mandatory) as well as secured (optional, defined project specific).⌋

[CP_SWS_VdpCmR_00154] Unsecured Tx PDU configuration

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The `VdpCmR` shall use the general purpose PDU defined by the mandatory container *VdpCmRCslTxPdu* for transmitting unsecured PDUs using the PduR.⌋

[CP_SWS_VdpCmR_00155] Secured Tx PDU configuration

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈If the optional container *VdpCmRCslSecuredTxPdu* is configured the VdpCmR shall use the general purpose PDU defined by this container for transmitting secured PDUs using the PduR.⌋

For VdpCmR there is no difference in handling unsecured and secured transmit PDUs. Depending on the chosen PDU ID the PduR will either forward the message to SecOC first or send it directly.

The user of VdpCmR has to decide if unsecured communication is sufficient or if secured communication is required for the use case. Unsecured communication may have drawbacks regarding security, secured communication the overhead for calculating the security parts. The VPD protocol allows defining secured transmission of the collected data on data point level by parameter *(SECOC)* inside the VDP settings of the data point dynamic configuration.

7.3 Processing of VDP Messages

The VdpCmR module receives VDP messages from the PduR via the Rx Data Path. These messages control VdpCmR as well as the connected DCA Remotes (realized as separate CDD in AUTOSAR).

[CP_SWS_VdpCmR_00101] Incoming Messages

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The VdpCmR shall provide the functionality to receive VDP messages from one other ECU via the PduR module.⌋

[CP_SWS_VdpCmR_00102] Receive buffer

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The VdpCmR shall use a receive buffer of size *VdpCmRRxBufferSize* to store received VDP messages.⌋

[CP_SWS_VdpCmR_00103] Receive StartOfReception

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The VdpCmR shall return the receive buffer to the PduR when this calls VdpCmR_StartOfReception (see Figure 9.1).⌋

[CP_SWS_VdpCmR_00104] Receive CopyRxData

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The VdpCmR shall copy the passed receive data into its receive buffer when the PduR calls VdpCmR_CopyRxData.⌋

[CP_SWS_VdpCmR_00105] Receive TpRxIndication

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The VdpCmR shall trigger processing the receive buffer in the next call to VdpCmR_MainFunction when the PduR calls VdpCmR_TpRxIndication.⌋

[CP_SWS_VdpCmR_00106] Receive buffer processing

Upstream requirements: FO_RS_VDP_00005, FO_RS_VDP_00006

⌈The VdpCmR shall parse and process the VDP messages inside the receive buffer inside VdpCmR_MainFunction when processing the receive buffer is requested and clear the receive buffer after processing.⌋

[CP_SWS_VdpCmR_00107] Incoming VDP Requests

Upstream requirements: FO_RS_VDP_00001, FO_RS_VDP_00002

⌈The module VdpCmR shall process incoming messages of type

- Get protocol version requests (*MT* = 0)
- Control message requests (*MT* = 1)

as specified in the protocol specification. All other values of *MT* shall be responded to with a protocol error message with *PEC* = 4.⌋

[CP_SWS_VdpCmR_00108] Get protocol version

Upstream requirements: FO_RS_VDP_00001

⌈The module VdpCmR shall process incoming *Get protocol version requests* by writing the protocol version response to the data message buffer and shall request sending the data message buffer in the next call to VdpCmR_MainFunction.⌋

[CP_SWS_VdpCmR_00109] Control message types

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00006

⌈The module VdpCmR shall process incoming control messages with the *command types (CT)*

- Add dynamic configuration (*CT* = 0)
- Remove dynamic configuration (*CT* = 1)
- Activation (*CT* = 2)
- Trigger (*CT* = 3)

as specified in the protocol specification. All other values of *CT* shall be responded to with a protocol error message with *PEC* = 1 (invalid options).⌋

[CP_SWS_VdpCmR_00119] Maximum VDP Slot ID

Upstream requirements: FO_RS_VDP_00009, FO_RS_VDP_00010

⌈The VdpCmR shall ignore a *VDP_SLOT_ID* that is bigger than the configured *VdpCmRVdpSlotIdCount*, i.e. the maximum valid *VDP_SLOT_ID*. In this case the VdpCmR shall add one error *VDP Slot ID out of range* (EC) = 0x77 to the data message buffer for the smallest of these *VDP_SLOT_ID*s of one request and request sending the data message buffer in the next call to VdpCmR_MainFunction.⌋

7.3.1 DCA Configuration

The VdpCmR interacts with the DCA Remotes, i.e. it configures them and collects the data provided by them.

[CP_SWS_VdpCmR_00210] DCA Remote count

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall communicate with the DCA Remotes configured inside container *VdpCmRDcaRemotes*. The number of the containers *VdpCmRDcaRemote* inside *VdpCmRDcaRemotes* define the number of DCA Remotes.⌋

[CP_SWS_VdpCmR_00211] DCA Remote names

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall use the API of a DCA Remote to communicate with it as described in chapter 8.6.3. The prefix of the function names <DcaRemote>_functionXXX (like <DcaRemote>_AddDpDCfg) is configured through the name of the enclosing container *VdpCmRDcaRemote*, the <DcaRemote> shall be replaced by this.⌋

[CP_SWS_VdpCmR_00212] DCA Remote ID mapping

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall realize a mapping between a DCA Remote ID and the corresponding DCA Remote by the configuration parameter *VdpCmRDcaRemoteId* that is contained inside the *VdpCmRDcaRemote* container. This allows calling the correct DCA Remote function when receiving a VDP message from the PduR containing just a DCA Remote ID.⌋

[CP_SWS_VdpCmR_00213] DCA Remote ID unknown

Upstream requirements: FO_RS_VDP_00010

⌈If the VdpCmR receives a control message containing an unknown DCA Remote ID it shall append an error *DCA Remote ID unknown* (*EC* = 0x76) together with the DCA Remote ID to the data message buffer and request sending the data message buffer in the next call to VdpCmR_MainFunction.⌋

7.3.2 Add dynamic configuration

If no payload was given in the request, the VdpCmR can respond almost immediately, previously caring about two error cases (a missing *TCT* or an already set *TCT*) as described in the protocol specification.

[CP_SWS_VdpCmR_00110] Add Dynamic Configuration - only cyclic transmission

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00011

⌈On reception of an add dynamic configuration request (*CT* = 0) with the flag *T_CYCLIC*, the VdpCmR shall validate that no cyclic transmission is currently configured and then start transmitting data messages regularly if *TCT* was given in the extended header.

In case, the request has no further payload, the VdpCmR shall immediately respond with an acknowledge.⌋

The following requirements specify the behavior of the VdpCmR module in interaction with the DCA Remotes.

[CP_SWS_VdpCmR_00111] Add Dynamic Configuration - DCA remote dynamic configuration

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00011

⌈In case the payload of the add dynamic configuration request (*CT* = 0) is not empty, the VdpCmR shall iterate over all *DCA_REM_DCFG* and for each *DCA_REM_DCFG* it shall perform the following steps to distribute the information to the correct DCA Remote

1. Find the DCA Remote which is indexed by the given DCA Remote ID.
2. Iterate over *DCA_REM_DP_DCFG_CT* following data point dynamic configurations *DP_DCFG*.
   (a) Build a VdpCmR_Dca_VdpSlotIdDpDCfgPairType from the information contained in *DP_DCFG*, and
   (b) Call the API <DcaRemote>_AddDpDCfg on the identified DCA Remote with the constructed VdpCmR_Dca_VdpSlotIdDpDCfgPairType.

⌋

In the following requirement the word "construct" shall indicate that another requirement is following describing the exact mapping. In case the verb "set" or "contain" is used, the value from the parameter on protocol level can be directly set into the programmatic parameter.

[CP_SWS_VdpCmR_00112] Add Dynamic Configuration - Construction of VdpCmR_Dca_VdpSlotIdDpDCfgPairType

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004

⌈To construct a `VdpCmR_Dca_VdpSlotIdDpDCfgPairType` from the information contained in *DP_DCFG*, the `VdpCmR` shall map the elements from the *DP_DCFG* as follows:

- the *VdpSlotId* in `VdpCmR_Dca_VdpSlotIdDpDCfgPairType` shall be set to *VDP_SLOT_ID*,
- information from *COL* in *DP_DCFG* shall be used to construct the *SamplingMode* of the `VdpCmR_Dca_VdpSlotIdDpDCfgPairType`,
- if given, the *SamplingCycleTime* of the `VdpCmR_Dca_VdpSlotIdDpDCfgPairType` shall be set to *SCT*, if not given, the *SamplingCycleTime* shall be set to 0,
- *INIT_ACT* from the Vdp settings *SET* shall be used to construct the *ActiveState* of the `VdpCmR_Dca_VdpSlotIdDpDCfgPairType`,
- the *DcaRemDpDCfgLen* of the `VdpCmR_Dca_VdpSlotIdDpDCfgPairType` shall be set to *DCA_REM_DP_DCFG_LEN*, and
- the *DcaRemDpDCfg* of the `VdpCmR_Dca_VdpSlotIdDpDCfgPairType` shall point to memory containing *DCA_REM_DP_DCFG*.

⌋

[CP_SWS_VdpCmR_00113] Add Dynamic Configuration - Sampling Mode

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003

⌈To construct a *SamplingMode* of the `VdpCmR_Dca_VdpSlotIdDpDCfgPairType` it shall be set based on *COL* on the protocol level as follows

- if *SCHANGE* = 0 and *SCYCLIC* = 0, *SamplingMode* = `VDPCMR_DCA_SAMPLING_MODE_ON_REQUEST`,
- if *SCHANGE* = 1 and *SCYCLIC* = 0, *SamplingMode* = `VDPCMR_DCA_SAMPLING_MODE_ON_CHANGE`,
- if *SCHANGE* = 0 and *SCYCLIC* = 1, *SamplingMode* = `VDPCMR_DCA_SAMPLING_MODE_CYCLIC`,
- if *SCHANGE* = 1 and *SCYCLIC* = 1, *SamplingMode* = `VDPCMR_DCA_SAMPLING_MODE_CYCLIC_OR_ON_CHANGE`.

⌋

[CP_SWS_VdpCmR_00114] Add Dynamic Configuration - Sampling Mode

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00004

⌈To construct a *ActiveState* of the `VdpCmR_Dca_VdpSlotIdDpDCfgPairType` it shall be set based on *SET* on the protocol level as follows

- if *INIT_ACT* = 0, *ActiveState* = `VDPCMR_DCA_DP_INACTIVE`,
- if *INIT_ACT* = 1, *ActiveState* = `VDPCMR_DCA_DP_ACTIVE`.

⌋

Beyond this, the `VdpCmR` needs to handle parts of the configuration internally as given in the following requirement.

[CP_SWS_VdpCmR_00115] Add Dynamic Configuration - VDP Settings

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00006, FO_RS_VDP_00010

⌈The `VdpCmR` shall implement processing of the VDP Settings except for the initial activation state given in each *DP_DCFG* and implement the according behavior on e.g. timestamp resolution and persistency in the module.⌋

[CP_SWS_VdpCmR_00116] Add Dynamic Configuration - Returned errors

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00010

⌈In case the call of the `VdpCmR` to the API `<DcaRemote>_AddDpDCfg` was not successful, i.e. did not return `E_OK`, the `VdpCmR` shall append a pair consisting of error code and concerned *VDP_SLOT_ID* to the add dynamic configuration response. The error codes from the API shall be mapped to error codes on protocol level as follows

- `VDPCMR_DCA_E_VDP_SLOT_ID_ALREADY_IN_USE` maps to *EC* = 0x7A (inconsistency error)
- `VDPCMR_DCA_E_INVALID_DCA_REM_DP_DCFG` maps to *EC* = 0x04
- `VDPCMR_DCA_E_DUPLICATE_DCA_REM_DP_DCFG` maps to *EC* = 0x05
- `VDPCMR_DCA_E_DP_LIMIT_REACHED` maps to *EC* = 0x06
- `VDPCMR_DCA_E_SAMPLING_MODE_CYCLIC_NOT_SUPPORTED` maps to *EC* = 0x07
- `VDPCMR_DCA_E_SAMPLING_MODE_ON_CHANGE_NOT_SUPPORTED` maps to *EC* = 0x08
- `VDPCMR_DCA_E_SAMPLING_MODE_CYCLIC_AND_ON_CHANGE_NOT_SUPPORTED` maps to *EC* = 0x09
- `E_NOT_OK` maps to *EC* = 0x00

⌋

Implementation specific error codes (i.e. 0x40-0x6F) are not mentioned explicitly as the further processing in the VdpCmR is allowed but not standardized.

7.3.3 Remove dynamic configuration

[CP_SWS_VdpCmR_00120] Remove Dynamic Configuration Request

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00011

⌈The module VdpCmR shall verify the combinations of the flags *(T_CYCLIC)*, *(GLOBAL)* and *(DCA)* together with the payload on validity and react on invalid combinations with an error according to the protocol specification.⌋

[CP_SWS_VdpCmR_00121] Remove Dynamic Configuration Request - cyclic

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004

⌈In case the *(T_CYCLIC)* flag is set, VdpCmR shall remove its internal cyclic transmission settings.⌋

[CP_SWS_VdpCmR_00122] Remove Dynamic Configuration Request - VDP Slot ID

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004

⌈In case no flag or the *(T_CYCLIC)* flag is set, VdpCmR shall iterate through the *VDP_SLOT_ID*s contained in the payload, find the corresponding DCA Remote and call <DcaRemote>_DeleteDpDCfg with the *VDP_SLOT_ID*.⌋

[CP_SWS_VdpCmR_00123] Remove Dynamic Configuration Request - Global

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004

⌈In case the *(GLOBAL)* flag is set, VdpCmR shall call <DcaRemote>_DeleteAllDpDCfgs for all existing DCAs.⌋

[CP_SWS_VdpCmR_00124] Remove Dynamic Configuration Request - DCA

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004

⌈In case the *(DCA)* flag is set, VdpCmR shall iterate through the DCAs contained inside the payload and call <DcaRemote>_DeleteAllDpDCfgs for each.⌋

7.3.4 Activation

[CP_SWS_VdpCmR_00130] Activation state

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00006

⌈The module VdpCmR shall activate the sampling in case the *activation state (ACT)* is set to 1 inside the activation request message or deactivate the sampling otherwise.⌋

[CP_SWS_VdpCmR_00131] Activation request payload

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00006

⌈The module VdpCmR shall decode all *VDP_SLOT_ID*s contained DDLE encoded inside the control message payload.⌋

[CP_SWS_VdpCmR_00132] Activation at remote DCA

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00006

⌈On reception of an activation request (*CT* = 2) with payload, the VdpCmR shall iterate through the provided *VDP_SLOT_ID*s and

1. validate the *VDP_SLOT_ID*, and
2. if successful, call the API <DcaRemote>_SetDpActivationState on the according DCA Remote with the *VDP_SLOT_ID* and the *activation state (ACT)* of the message as parameter.

⌋

[CP_SWS_VdpCmR_00133] Activation request - Validation in VdpCmR

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00011

⌈If the validation of a *VDP_SLOT_ID* in the processing of the activation request was not successful, the VdpCmR shall append to the payload of the response the suitable error code *EC* (one of the ones allowed in the protocol specification for the activation response) and the concerned *VDP_SLOT_ID*.

The VdpCmR shall also set (if not yet set), the acknowledge flag *ACK* in the response to 0 (not acknowledge).⌋

Here, resolving the "according DCA Remote" needs to be executed in the VdpCmR. This is possible by a simple look-up table which is edited when add dynamic configuration or remove dynamic configuration requests are processed.

[CP_SWS_VdpCmR_00134] Activation request - Sampling Error in DCA Remote

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00011

⌈If the call to <DcaRemote>_SetDpActivationState with a specific *VDP_SLOT_ID* returns something different than E_OK, the VdpCmR shall append to the payload of the activation response the following mapped error code

- E_NOT_OK : *EC* = 0x00 (generic error)
- VDPCMR_DCA_E_UNKNOWN_VDP_SLOT_ID : *EC* = 0x7A (inconsistency error)
- 0x40 – 0x6F : *EC* is equal to the return value.

and the *VDP_SLOT_ID*.

The VdpCmR shall also set (if not yet set), the acknowledge flag *ACK* in the response to 0 (not acknowledge).⌋

[CP_SWS_VdpCmR_00135] Activation request - Response

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00011

⌈The VdpCmR shall respond only to the activation request after the whole payload of the request has been processed, as well as the *ACT* flag, and the response payload was built (if necessary) containing all errors which occurred.⌋

7.3.5 Trigger

[CP_SWS_VdpCmR_00140] Trigger request - Transmission

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004

⌈On reception of a trigger request (*CT* = 3) with *TX_TRIG* = 1, the VdpCmR shall request sending the data message buffer in the next call to VdpCmR_MainFunction.

If no payload is contained, the VdpCmR shall respond immediately with an acknowledge (*ACK* = 1) with no payload.⌋

The payload in trigger requests is optional. If contained the processing shall be done according to the following requirement.

[CP_SWS_VdpCmR_00141] Trigger request - Sampling

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004

⌈On reception of a trigger request (*CT* = 3) with payload, the VdpCmR shall iterate through the provided *VDP_SLOT_ID*s and

1. validate the *VDP_SLOT_ID*, and
2. if successful, call the API <DcaRemote>_TriggerDpOneTimeSampling on the according DCA Remote with the *VDP_SLOT_ID* as parameter.

⌋

In the previous requirement, it is not required that the sampling must occur immediately during the API call to <DcaRemote>_TriggerDpOneTimeSampling. It is allowed for the DCA Remote to sample later in e.g. the next main function cycle.

The allowed error codes and their usage is sufficiently described in the protocol specification of VDP, hence, only the fact that an error code is appended remains to be specified.

[CP_SWS_VdpCmR_00142] Trigger request - Sampling - Validation in VdpCmR

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00005

⌈If the validation of a *VDP_SLOT_ID* in the processing of the trigger request was not successful, the VdpCmR shall append to the payload of the response the suitable error code *EC* (one of the ones allowed in the protocol specification for the trigger response) and the concerned *VDP_SLOT_ID*.

The VdpCmR shall also set (if not yet set), the acknowledge flag *ACK* in the response to 0 (not acknowledge).⌋

Here, resolving the "according DCA Remote" needs to be executed in the VdpCmR. This is possible by a simple look-up table which is edited when add dynamic configuration or remove dynamic configuration requests are processed.

[CP_SWS_VdpCmR_00143] Trigger request - Sampling Error in DCA Remote

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00005, FO_RS_VDP_00011

⌈If the call to <DcaRemote>_TriggerDpOneTimeSampling with a specific *VDP_SLOT_ID* returns something different than E_OK, the VdpCmR shall append to the payload of the trigger response the following mapped error code

- E_NOT_OK : *EC* = 0x00 (generic error)
- VDPCMR_DCA_E_SAMPLING_ERROR : *EC* = 0x02 (sampling could not be triggered)
- VDPCMR_DCA_E_UNKNOWN_VDP_SLOT_ID : *EC* = 0x7A (inconsistency error)
- 0x40 – 0x6F : *EC* is equal to the return value.

and the *VDP_SLOT_ID*.

The VdpCmR shall also set (if not yet set), the acknowledge flag *ACK* in the response to 0 (not acknowledge).⌋

[CP_SWS_VdpCmR_00144] Trigger request - Response

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003, FO_RS_VDP_00004, FO_RS_VDP_00011

⌈The VdpCmR shall respond only to the trigger request after the whole payload of the request has been processed, as well as the *TX_TRIG* flag, and the response payload was built (if necessary) containing all errors which occurred.⌋

7.4 Submission of Data Point Samples

When a DCA Remote calls VdpCmR_AddSample, the VdpCmR stores the provided data suitably inside its data message buffer of configuration parameter size *VdpCmRTxBufferSize*. Beyond this, the function contains no further functionality unless the transmission is configured to be "on sample" for this specific data point.

[CP_SWS_VdpCmR_00401] Adding data point samples

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00009

⌈The VdpCmR shall be able to receive data point samples with the service VdpCmR_AddSample.

If all parameters are valid, the VdpCmR shall then add a new *Data Point Sample Information (DPSI)* with the *vdpslotid* from the API call in the *VDP_SLOT_ID* of the *DPSI* to the data message buffer.

When this is successful, the VdpCmR shall return E_OK.⌋

[CP_SWS_VdpCmR_00402] Adding too big sample

Upstream requirements: FO_RS_VDP_00010

⌈The VdpCmR shall return E_NOT_OK if a data message is passed by VdpCmR_AddSample that is bigger than *VdpCmRMaxDataLen*.⌋

Argument invalidity to VdpCmR_AddSample is mostly explained in [CP_SWS_VdpCmR_01203].

[CP_SWS_VdpCmR_00410] Get current timestamp

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00009

⌈When VdpCmR_AddSample is called, VdpCmR shall retrieve the current timestamp by StbM_GetCurrentTime and time base identifier *VdpCmRStbMSynchronizedTimeBaseRef*.⌋

[CP_SWS_VdpCmR_00406] First data sample in message

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00009

⌈When VdpCmR_AddSample is called and it is the first sample after the transmission of the last data message, the reference timestamp *REF_TS* of the next data message shall be set to the current timestamp and the relative timestamp *REF_TS* in the *DPSI* shall be set to 0.⌋

[CP_SWS_VdpCmR_00403] Consecutive data sample in message

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00009

⌈When VdpCmR_AddSample is called and it is not the first sample after the transmission of the last data message, the relative timestamp *REF_TS* shall be computed with respect to the timestamp of the previous sample in the requested resolution.⌋

[CP_SWS_VdpCmR_00409] Timebase error

Upstream requirements: FO_RS_VDP_00010

⌈When a timestamp used inside VdpCmR_AddSample is out of sync (determinable through timestamp element *timeBaseStatus*), VdpCmR shall write a *timebase error* (EC) = 0x72 into the data message buffer before writing the data point. The *timebase error* shall be written only once to each data message buffer.⌋

It is important that the difference computation is based on absolute but not relative timestamps.

[CP_SWS_VdpCmR_00404] Data

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00009

⌈When VdpCmR_AddSample is called, the data length *DLEN* in the *DPSI* shall be set to the DDLE encoded value of *dataLen* and *dataLen* bytes shall be read from ptrData and inserted into the *DATA* field of *DPSI*.⌋

Transmission of the data samples shall always happen asynchronously in the VdpCmR_MainFunction. (see Figure 9.2)

[CP_SWS_VdpCmR_00405] On-sampling transmission

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00009

⌈In case a data point shall be transmitted on-sampling, the call to VdpCmR_AddSample shall request sending the data message buffer in the next call to VdpCmR_MainFunction.⌋

7.4.1 Error handling

Most potential errors reported to the DCA Remote are sufficiently specified in [CP_SWS_VdpCmR_01203]. In particular a non-successful call to VdpCmR_AddSample may be answered by VDPCMR_E_BUFFER_FULL, which shall also be used when the module internal data message buffer is full and needs to be transmitted before the call to VdpCmR_AddSample can be successful.

[CP_SWS_VdpCmR_00407] AddSample Buffer full

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00009, FO_RS_VDP_00010

⌈The VdpCmR shall always reserve space in the data message buffer to append an asynchronous error *ASYNC_ERR* which indicates that the data message buffer is full, i.e. *(EC) = 0x74*. If the remaining buffer space is not sufficient to store the next *DPSI* for the data sample provided with VdpCmR_AddSample, the VdpCmR shall append the previously mentioned asynchronous error to the data message buffer and request sending it in the next call to VdpCmR_MainFunction.

The VdpCmR_AddSample shall then return VDPCMR_E_BUFFER_FULL.⌋

[CP_SWS_VdpCmR_00408] Maximum timestamp length

Upstream requirements: FO_RS_VDP_00009, FO_RS_VDP_00010

⌈The VdpCmR shall limit the size of DDLE encoded timestamps to configuration parameter *VdpCmRMaxRelTimestampLen*. If the encoded timestamp would need more bytes to be stored VdpCmR shall set the timestamp to the maximum timestamp value (all bits set) and return E_NOT_OK.⌋

7.5 Submission of Asynchronous Errors

When a DCA Remote calls VdpCmR_AddAsyncError, the VdpCmR adds an asynchronous error to the data message buffer. These errors are transmitted within the data frame as a specific type of the data point information. In general, this function is similar to VdpCmR_AddSample except that instead of a data point error information is contained.

[CP_SWS_VdpCmR_00501] Adding asynchronous errors

Upstream requirements: FO_RS_VDP_00009, FO_RS_VDP_00010

⌈The VdpCmR shall be able to receive asynchronous error information with the service VdpCmR_AddAsyncError.

If all parameters are valid, the VdpCmR shall add a new *Asynchronous Error Information (ASYNC_ERR)* with the *VDP_SLOT_ID* set to the *vdpslotid* from the API call to the data message buffer.

When this is successful, the VdpCmR shall return E_OK⌋

Argument invalidity to VdpCmR_AddAsyncError is mostly explained in [CP_SWS_VdpCmR_01204].

[CP_SWS_VdpCmR_00503] Error Information

Upstream requirements: FO_RS_VDP_00010

⌈When VdpCmR_AddAsyncError is called, the error information length *ERR_LEN* in the *ASYNC_ERR* shall be set to the value of *errorInfoLen* and *errorInfoLen* bytes shall be read from *ptrErrorInfo* and inserted into the *ERR_INFO* field of *ASYNC_ERR*.⌋

[CP_SWS_VdpCmR_00504] Asynchronous Error already contained

Upstream requirements: FO_RS_VDP_00010

⌈When VdpCmR_AddAsyncError is called with an error, i.e. the pair *VDP_SLOT_ID* and *errorCode*, that is already contained inside the data message buffer it will not add this error a second time and return E_OK.⌋

[CP_SWS_VdpCmR_00505] Data Source Error

Upstream requirements: FO_RS_VDP_00010

⌈When VdpCmR_AddAsyncError is called with *errorCode* Data source error, DCA Remote shall write the DDLE encoded DCA Remote ID into *ERR_INFO*.⌋

Transmission of the asynchronous errors shall always happen similar to the data samples asynchronously in the VdpCmR_MainFunction.

7.5.1 Error handling

Most potential errors reported to the DCA Remote are sufficiently specified in [CP_SWS_VdpCmR_01204]. In particular a non-successful call to VdpCmR_AddAsyncError may be answered by VDPCMR_E_BUFFER_FULL, which shall also be used when the module internal data message buffer is full and needs to be transmitted before the call to VdpCmR_AddAsyncError can be successful.

[CP_SWS_VdpCmR_00506] AddAsyncError Buffer full

Upstream requirements: FO_RS_VDP_00009

⌈The VdpCmR shall always reserve space in the next outgoing data message buffer to append an asynchronous error *ASYNC_ERR* which indicates that the buffer is full, i.e. *(EC) = 0x74*. If the remaining buffer space is not sufficient to store the next *ASYNC_ERR* for the error provided with VdpCmR_AddAsyncError, the VdpCmR shall append the previously mentioned asynchronous error to the data message buffer and request sending it in the next call to VdpCmR_MainFunction.

The VdpCmR_AddAsyncError shall then return VDPCMR_E_BUFFER_FULL.⌋

7.6 Transmission of Data Messages

This section describes the transmission of data messages, which is only executed in VdpCmR_MainFunction. See also [CP_SWS_VdpCmR_00405] in the previous section for an explanation how the transmission mode on-sampling shall be realized.

[CP_SWS_VdpCmR_00701] Cyclic call VdpCmR_MainFunction

Upstream requirements: FO_RS_VDP_00004

⌈The VdpCmR function VdpCmR_MainFunction shall be called cyclic with the configured period *VdpCmRMainFunctionPeriod*.⌋

[CP_SWS_VdpCmR_00702] Cyclic transmission

Upstream requirements: FO_RS_VDP_00004

⌈In case the transmission mode cyclic is set, the VdpCmR shall validate on call of VdpCmR_MainFunction, whether the cycle time is expired. If so, VdpCmR shall request sending the data message in this call of VdpCmR_MainFunction.⌋

[CP_SWS_VdpCmR_00703] Buffer threshold exceeded

Upstream requirements: FO_RS_VDP_00009

⌈When `VdpCmR_MainFunction` is called, the `VdpCmR` shall check whether a compile time configured threshold (*VdpCmRTxAutoTransmissionThreshold*) of the data message buffer is already filled with data point samples. If so, `VdpCmR` shall request sending the data message buffer in this call of `VdpCmR_MainFunction`.⌋

[CP_SWS_VdpCmR_00704] Transmission and MTDT

Upstream requirements: FO_RS_VDP_00009

⌈In `VdpCmR_MainFunction`, after checking the transmission mode cyclic and the buffer threshold, the `VdpCmR` shall check if sending the data message is requested. If so, the `VdpCmR` shall validate that after the last transmission of a data message at least the minimum transmission distance time (MTDT) configured through *VdpCmRMinTxDistanceTime* is reached. If so, the `VdpCmR` shall lock the data message buffer and call `PduR_VdpCmRTransmit` with the data message buffer containing all data samples and asynchronous errors since the last transmission. (see Figure 9.2) Otherwise, `VdpCmR` shall postpone sending the data message to the next call of `VdpCmR_MainFunction`.⌋

[CP_SWS_VdpCmR_00705] Data Message Buffer Locked

Upstream requirements: FO_RS_VDP_00004

⌈If the data message buffer is locked the `VdpCmR` shall respond with error `VDPCMR_E_BUFFER_BUSY` to any call of `VdpCmR_AddSample` or `VdpCmR_AddAsyncError`.⌋

[CP_SWS_VdpCmR_00706] Transmission copy data

Upstream requirements: FO_RS_VDP_00004

⌈After starting the transmission with `PduR_VdpCmRTransmit` the PduR will call `VdpCmR_CopyTxData` one or multiple times. The `VdpCmR` shall copy the data message buffer into the provided PDU buffer. (see Figure 9.2)⌋

[CP_SWS_VdpCmR_00707] Transmission confirmation

Upstream requirements: FO_RS_VDP_00004

⌈When the PduR calls `VdpCmR_TpTxConfirmation` for the transmit PDU, `VdpCmR` shall clear the data message buffer and release its lock independent of the result of `VdpCmR_TpTxConfirmation`.⌋

`VdpCmR` does not realize any retries in error case or quality of service mechanisms. On receiver side it is possible to determine a data packet loss through the sequence counter.

7.7 Persistency of Dynamic Configurations

VdpCmR might persist the Dynamic Configuration of data points to be able to start immediately with data collection after start-up, e.g. after a reset. It will not persist collected data.

[CP_SWS_VdpCmR_00601] Persistency

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall be able to conditionally persist the *DP_DCFG* in the NvM.⌋

[CP_SWS_VdpCmR_00602] NvM Block Descriptor

Upstream requirements: FO_RS_VDP_00002

⌈The NvM block descriptor referenced by ECUC parameter *VdpCmRNvmBlockDescriptor*, shall be a block processed during NvM_ReadBlock, NvM_WriteBlock, NvM_GetErrorStatus or NvM_SetRamBlockStatus.⌋

[CP_SWS_VdpCmR_00603] NvM Block Name

Upstream requirements: FO_RS_VDP_00002

⌈The supported NvM block name shall be VdpCmR_NvmRomBlockData.⌋

[CP_SWS_VdpCmR_00604] Read Persistent Data

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall use the function NvM_ReadBlock to read the persistent VdpCmR data block from the NvM.⌋

[CP_SWS_VdpCmR_00605] Write Persistent Data

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall use the function NvM_WriteBlock to write the persistent VdpCmR data block to the NvM.⌋

[CP_SWS_VdpCmR_00606] Get Persistent Data Error Status

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall use the function NvM_GetErrorStatus to get the NvM block dependent status and error information.⌋

[CP_SWS_VdpCmR_00607] NvM RAM Block Mirror

Upstream requirements: FO_RS_VDP_00002

⌈The VdpCmR shall use the function NvM_SetRamBlockStatus to update the RAM mirror status of the persistent VdpCmR data block in case of using a RAM mirror.⌋

Using a RAM mirror is optional. It may decouple the data access from slow access to the NvM. Additionally, it could improve the durability of the NvM by reducing the number of write accesses.

[CP_SWS_VdpCmR_00608] Enable Persistence

Upstream requirements: FO_RS_VDP_00002

⌈In case ECUC parameter *VdpCmRNvmBlockDescriptor* is configured, the persistency is enabled for `VdpCmR`. Then, it shall persist each *DP_DCFG* where *(PERSIST)* = 1 is configured.⌋

[CP_SWS_VdpCmR_00609] Load Persistent Configuration

Upstream requirements: FO_RS_VDP_00002

⌈If persistency is enabled, inside function `VdpCmR_Init` the `VdpCmR` shall load the persisted *DP_DCFG* entries and configure the corresponding `DCA Remotes` similar to an *add dynamic configuration request*. (see chapter 7.3.2)⌋

[CP_SWS_VdpCmR_00610] Load Persistent Configuration Fails

Upstream requirements: FO_RS_VDP_00002

⌈If loading the persisted configuration fails `VdpCmR` shall start without any *DP_DCFG* (similar to the persistency disabled case).⌋

[CP_SWS_VdpCmR_00611] Store Persistent Configuration

Upstream requirements: FO_RS_VDP_00002

⌈If persistency is enabled, the `VdpCmR` shall write the *DP_DCFG*s with *(PERSIST)* = 1 to the NvM after processing the *add dynamic configuration request* and/or cyclic inside `VdpCmR_MainFunction`.⌋

7.8 Error Classification

Section "Error Handling" of the document [3] "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.8.1 Development Errors

[CP_SWS_VdpCmR_01000] Definition of development errors in module VdpCmR

Upstream requirements: SRS_BSW_00337

⌈

Type of error	*Related error code*	*Error value*
Component is uninitialized	VDPCMR_E_UNINIT	0x1
Passed argument was invalid	VDPCMR_E_ARG_INVALID	0x2
Length of data exceeds pre-configured data length	VDPCMR_E_DATA_LEN_EXCEEDED	0x3

⌋

[CP_SWS_VdpCmR_01001] VdpCmR Error Not Initialized

Upstream requirements: SRS_BSW_00337, SRS_BSW_00369

⌈If development error detection is enabled for the `VdpCmR` module, any function call of the public API except `VdpCmR_Init` and `VdpCmR_GetVersionInfo` shall report the DET error `VDPCMR_E_UNINIT` when `VdpCmR` is not yet initialized.⌋

[CP_SWS_VdpCmR_01002] VdpCmR Error Invalid Argument

Upstream requirements: SRS_BSW_00337, SRS_BSW_00369

⌈If development error detection is enabled for the `VdpCmR` module, any function call of the public API shall report the DET error `VDPCMR_E_ARG_INVALID` when `VdpCmR` is called with an invalid argument, e.g. a NULL pointer.⌋

[CP_SWS_VdpCmR_01003] VdpCmR Error Data Length Exceeded

Upstream requirements: SRS_BSW_00337, SRS_BSW_00369

⌈If development error detection is enabled for the `VdpCmR` module, any call of `VdpCmR_AddSample` shall report the DET error `VDPCMR_E_DATA_LEN_EXCEEDED` when the passed data is bigger than the pre-configured data length *VdpCmRMaxDataLen*.⌋

7.8.2 Runtime Errors

There are no runtime errors.

7.8.3 Production Errors

There are no production errors.

7.8.4 Extended Production Errors

There are no extended production errors.

7.9 Security Events

The module does not report security events.

8 API specification

8.1 Imported types

In this chapter all types included by the VdpCmR are listed together with the file and the defining module.

[CP_SWS_VdpCmR_02000] Definition of imported datatypes of module VdpCmR

Upstream requirements: SRS_BSW_00301

⌈

Module	*Header File*	*Imported Type*
Comtype	ComStack_Types.h	BufReq_ReturnType
	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
	ComStack_Types.h	RetryInfoType
	ComStack_Types.h	TpDataStateType
NvM	Rte_NvM_Type.h	NvM_BlockIdType
	Rte_NvM_Type.h	NvM_RequestResultType
StbM	Rte_StbM_Type.h	StbM_SynchronizedTimeBaseType
	Rte_StbM_Type.h	StbM_TimeBaseStatusType
	Rte_StbM_Type.h	StbM_TimeStampType
	Rte_StbM_Type.h	StbM_TimeTupleType
	Rte_StbM_Type.h	StbM_UserDataType
	StbM.h	StbM_VirtualLocalTimeType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋

8.2 Type definitions

8.2.1 VdpCmR_ReturnType

[CP_SWS_VdpCmR_01100] Definition of Std_ReturnType-extension for module VdpCmR

Upstream requirements: SRS_BSW_00331

⌈

Range	VDPCMR_DCA_E_UNKNOWN_VDP_SLOT_ID	0x02	Unknown VDP Slot ID.
	VDPCMR_DCA_E_VDP_SLOT_ID_ALREADY_IN_USE	0x03	VDP Slot ID already in use.
	VDPCMR_DCA_E_SAMPLING_ERROR	0x04	Sampling of the data point has failed.
	VDPCMR_DCA_E_INVALID_DCA_REM_DP_DCFG	0x05	The provided DCA_REM_DP_DCFG is invalid.
	VDPCMR_DCA_E_DUPLICATE_DCA_REM_DP_DCFG	0x06	The provided DCA_REM_DP_DCFG is already submitted.
	VDPCMR_DCA_E_DP_LIMIT_REACHED	0x07	The maximum amount of configurable data points for this DCA Remote is reached.
	VDPCMR_DCA_E_SAMPLING_MODE_CYCLIC_NOT_SUPPORTED	0x08	Sampling Mode Cyclic not supported.
	VDPCMR_DCA_E_SAMPLING_MODE_ON_CHANGE_NOT_SUPPORTED	0x09	Sampling mode on change not supported.
	VDPCMR_DCA_E_SAMPLING_MODE_CYCLIC_AND_ON_CHANGE_NOT_SUPPORTED	0x0A	Sampling mode cyclic and on change not supported.
	VDPCMR_DCA_E_CUSTOM_ERROR_0	0x40	First of 48 custom errors, can be defined from the implementer of the DCA.
	VDPCMR_DCA_E_CUSTOM_ERROR_47	0x6F	Last of 48 custom errors, can be defined from the implementer of the DCA.
	VDPCMR_E_BUFFER_FULL	0xA0	Buffer is full.
	VDPCMR_E_BUFFER_BUSY	0xA1	Buffer is currently being accessed.
	VDPCMR_E_RECONFIGURING	0xA2	Reconfiguration ongoing.
	VDPCMR_E_VDP_SLOT_ID_INVALID	0xA3	VDP Slot ID invalid.
Description	Overlayed return value of Std_ReturnType for the VdpCmR module.		

▽

△

Available via	VdpCmR.h

⌋

8.2.2 VdpCmR_ConfigType

[CP_SWS_VdpCmR_01101] Definition of datatype VdpCmR_ConfigType

Upstream requirements: SRS_BSW_00101, SRS_BSW_00344, SRS_BSW_00358, SRS_BSW_00404

⌈

Name	VdpCmR_ConfigType	
Kind	Structure	
Elements	Implementation specific	
	Type	–
	Comment	The content of the initialization data structure is implementation specific.
Description	This is the type of the data structure containing the initialization data for VdpCmR.	
Available via	VdpCmR.h	

⌋

Since this type is used for compliance purposes only (meaning that `VdpCmR_Init` will now have a pointer to this type as parameter, based on SWS_BSW_00050) it will be left to the developer to choose how to implement it, considering it has no use for the `VdpCmR` module in any way.

8.2.3 DCA Remote Types

This chapter contains the type definitions needed for the interaction of the `VdpCmR` with the `DCA Remotes`.

8.2.3.1 VdpCmR_Dca_DpActiveStateEnumType

[CP_SWS_VdpCmR_01102] Definition of datatype VdpCmR_Dca_DpActiveStateEnumType

Upstream requirements: FO_RS_VDP_00003

⌈

Name	VdpCmR_Dca_DpActiveStateEnumType		
Kind	Enumeration		
Range	VDPCMR_DCA_DP_INACTIVE	0x00	Data point shall currently not be sampled automatically.
	VDPCMR_DCA_DP_ACTIVE	0x01	Data point shall currently be sampled automatically.
Description	This enumeration contains the possible activation states of a data point.		
Available via	VdpCmR_Externals.h		

⌋

8.2.3.2 VdpCmR_Dca_SamplingModeEnumType

[CP_SWS_VdpCmR_01103] Definition of datatype VdpCmR_Dca_SamplingModeEnumType

Upstream requirements: FO_RS_VDP_00003

⌈

Name	VdpCmR_Dca_SamplingModeEnumType		
Kind	Enumeration		
Range	VDPCMR_DCA_SAMPLING_MODE_ON_REQUEST	0x00	Data is sampled through trigger requests (on-request).
	VDPCMR_DCA_SAMPLING_MODE_CYCLIC	0x01	Data is sampled cyclically or on-request.
	VDPCMR_DCA_SAMPLING_MODE_ON_CHANGE	0x02	Data is sampled on-change or on-request.
	VDPCMR_DCA_SAMPLING_MODE_CYCLIC_OR_ON_CHANGE	0x03	Data is sampled cyclically, on-change or on-request.
Description	This enumeration contains the possible sampling modes of a DCA Remote.		
Available via	VdpCmR_Externals.h		

⌋

8.2.3.3 VdpCmR_Dca_VdpSlotIdDpDCfgPairType

[CP_SWS_VdpCmR_01104] Definition of datatype VdpCmR_Dca_VdpSlotIdDpDCfgPairType

Upstream requirements: FO_RS_VDP_00002

⌈

Name	VdpCmR_Dca_VdpSlotIdDpDCfgPairType	
Kind	Structure	
Elements	DcaRemDpDCfg	
	Type	uint8*
	Comment	Pointer to the dynamic configuration for this data point.
	DcaRemDpDCfgLen	
	Type	uint16
	Comment	Length of the content in DcaRemDpDCfg
	SamplingCycleTime	
	Type	uint16
	Comment	Cycle time in milliseconds
	ActiveState	
	Type	VdpCmR_Dca_DpActiveStateEnumType
	Comment	Initial activation state of the data point.
	SamplingMode	
	Type	VdpCmR_Dca_SamplingModeEnumType
	Comment	Indicates how the sampling mode shall be acquired.
	VdpSlotId	
	Type	uint16
	Comment	VdpSlotId on which the samples can be submitted and with which activation and triggers can be set.
Description	Aggregates all necessary information to configure the sampling of the data point in the DCA Remote together with the VdpSlotId.	
Available via	VdpCmR_Externals.h	

⌋

8.3 Function definitions

8.3.1 VdpCmR_Init

[CP_SWS_VdpCmR_01200] Definition of API function VdpCmR_Init

Upstream requirements: SRS_BSW_00101, SRS_BSW_00344, SRS_BSW_00358, SRS_BSW_00404

⌈

Service Name	VdpCmR_Init	

Syntax	`void VdpCmR_Init (`
	`  const VdpCmR_ConfigType* configPtr`
	`)`	

Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	configPtr	Pointer to the selected configuration set.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the component internal tables and sets up internal buffers. Note: in case a reference to an Nv block is configured, the NvM needs to be initialized before the VdpCmR is initialized.	
Available via	VdpCmR.h	

⌋

[CP_SWS_VdpCmR_01201] VdpCmR ConfigPtr at Init

Upstream requirements: SRS_BSW_00414

⌈The Configuration pointer `ConfigPtr` is currently not used and shall therefore be set to the `NULL_PTR` value when calling the `VdpCmR_Init` API.⌋

8.3.2 VdpCmR_GetVersionInfo

[CP_SWS_VdpCmR_01202] Definition of API function VdpCmR_GetVersionInfo

Upstream requirements: SRS_BSW_00402

⌈

Service Name	VdpCmR_GetVersionInfo

Syntax	`void VdpCmR_GetVersionInfo (`
	`  Std_VersionInfoType* versionInfo`
	`)`

Service ID [hex]	0x01
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None

▽

△

Parameters (inout)	None	
Parameters (out)	versionInfo	Pointer to where to store the version information of this module
Return value	None	
Description	Returns the version information of this module.	
Available via	VdpCmR.h	

⌋

8.3.3 API for DCA Remote

This section contains functions of the VdpCmR that must be called by the DCA Remotes.

8.3.3.1 VdpCmR_AddSample

[CP_SWS_VdpCmR_01203] Definition of API function VdpCmR_AddSample

Upstream requirements: FO_RS_VDP_00002

⌈

Service Name	VdpCmR_AddSample	
Syntax	`Std_ReturnType VdpCmR_AddSample ( uint16 vdpSlotId, const void* ptrData, uint16 dataLen )`	
Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	vdpSlotId	VDP Slot ID assigned to the datapoint.
	ptrData	Pointer to the data sample's data.
	dataLen	Length of the data sample's data.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: No error occurred E_NOT_OK: Generic error: will be reported in any development error case or generic error VDPCMR_E_BUFFER_FULL: Buffer full, either drop data or retry VDPCMR_E_BUFFER_BUSY: Buffer is currently being accessed, retry VDPCMR_E_RECONFIGURING: Reconfiguration is ongoing, drop data VDPCMR_E_VDP_SLOT_ID_INVALID: VDP Slot ID invalid
Description	The DCA Remote uses this function to add a sample from a given data point to the data message buffer. The VdpCmR adds the current timestamp to the sample.	
Available via	VdpCmR.h	

⌋

8.3.3.2 VdpCmR_AddAsyncError

[CP_SWS_VdpCmR_01204] Definition of API function VdpCmR_AddAsyncError

Upstream requirements: FO_RS_VDP_00009, FO_RS_VDP_00010

⌈

Service Name	VdpCmR_AddAsyncError	
Syntax	`Std_ReturnType VdpCmR_AddAsyncError ( uint16 vdpSlotId, uint8 errorCode, const uint8* ptrErrorInfo, uint8 errorInfoLen )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	vdpSlotId	The VDP Slot ID assigned to the datapoint causing the error.
	errorCode	The asynchronous error code that shall be added.
	ptrErrorInfo	The error information associated to the error, may be a nullpointer.
	errorInfoLen	The length of the error information which shall be appended.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: No error occurred E_NOT_OK: will be reported in any development error case or generic error VDPCMR_E_BUFFER_FULL: Buffer full, either drop data or retry VDPCMR_E_BUFFER_BUSY: Buffer is currently being accessed, retry VDPCMR_E_RECONFIGURING: Reconfiguration is ongoing, drop data VDPCMR_E_VDP_SLOT_ID_INVALID: VDP Slot ID invalid
Description	The DCA Remote uses this function to add an asynchronous error to the data message buffer.	
Available via	VdpCmR.h	

⌋

8.4 Callback notifications

There are only callback notifications for the PduR.

8.4.1 API for PduR

This section contains callback notifications of the PduR. They are specified and called by PduR but implemented by the VdpCmR.

8.4.1.1 VdpCmR_StartOfReception

[CP_SWS_VdpCmR_01300] Definition of callback function VdpCmR_StartOfReception

Upstream requirements: FO_RS_VDP_00002

⌈

Service Name	VdpCmR_StartOfReception	
Syntax	`BufReq_ReturnType VdpCmR_StartOfReception ( PduIdType id, const PduInfoType* info, PduLengthType TpSduLength, PduLengthType* bufferSizePtr )`	
Service ID [hex]	0x46	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the I-PDU.
	info	Pointer to a PduInfoType structure containing the payload data (without protocol information) and payload length of the first frame or single frame of a transport protocol I-PDU reception, and the MetaData related to this PDU. If neither first/single frame data nor MetaData are available, this parameter is set to NULL_PTR.
	TpSduLength	Total length of the N-SDU to be received.
Parameters (inout)	None	
Parameters (out)	bufferSizePtr	Available receive buffer in the receiving module. This parameter will be used to compute the Block Size (BS) in the transport protocol module.
Return value	BufReq_ReturnType	`BUFREQ_OK`: Connection has been accepted. bufferSizePtr indicates the available receive buffer; reception is continued. If no buffer of the requested size is available, a receive buffer size of 0 shall be indicated by bufferSizePtr. `BUFREQ_E_NOT_OK`: Connection has been rejected; reception is aborted. bufferSizePtr remains unchanged. `BUFREQ_E_OVFL`: No buffer of the required length can be provided; reception is aborted. bufferSizePtr remains unchanged.
Description	The function call indicates the reception start of a segmented PDU.	
Available via	VdpCmR_Externals.h	

⌋

8.4.1.2 VdpCmR_CopyRxData

[CP_SWS_VdpCmR_01301] Definition of callback function VdpCmR_CopyRxData

Upstream requirements: FO_RS_VDP_00002

⌈

Service Name	VdpCmR_CopyRxData	
Syntax	`BufReq_ReturnType VdpCmR_CopyRxData ( PduIdType id, const PduInfoType* info, PduLengthType* bufferSizePtr )`	
Service ID [hex]	0x44	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the received I-PDU.
	info	Provides the source buffer (SduDataPtr) and the number of bytes to be copied (SduLength). An SduLength of 0 can be used to query the current amount of available buffer in the upper layer module. In this case, the SduDataPtr may be a NULL_PTR.
Parameters (inout)	None	
Parameters (out)	bufferSizePtr	Available receive buffer after data has been copied.
Return value	BufReq_ReturnType	`BUFREQ_OK`: Data copied successfully `BUFREQ_E_NOT_OK`: Data was not copied because an error occurred.
Description	This function is called to provide the received data of an I-PDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the I-PDU data. The size of the remaining buffer is written to the position indicated by bufferSizePtr.	
Available via	VdpCmR_Externals.h	

⌋

8.4.1.3 VdpCmR_TpRxIndication

[CP_SWS_VdpCmR_01302] Definition of callback function VdpCmR_TpRxIndication

Upstream requirements: FO_RS_VDP_00002

⌈

Service Name	VdpCmR_TpRxIndication	
Syntax	`void VdpCmR_TpRxIndication ( PduIdType id, Std_ReturnType result )`	
Service ID [hex]	0x45	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the received I-PDU.

▽

△

	result	E_OK: The PDU was received. E_NOT_OK: Reception of the PDU failed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Called after an I-PDU has been received via the TP API, the result indicates whether the transmission was successful or not.	
Available via	VdpCmR_Externals.h	

⌋

8.4.1.4 VdpCmR_CopyTxData

[CP_SWS_VdpCmR_01303] Definition of callback function VdpCmR_CopyTx Data

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00010

⌈

Service Name	VdpCmR_CopyTxData	
Syntax	`BufReq_ReturnType VdpCmR_CopyTxData ( PduIdType id, const PduInfoType* info, const RetryInfoType* retry, PduLengthType* availableDataPtr )`	
Service ID [hex]	0x43	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the transmitted I-PDU.
	info	Provides the destination buffer (SduDataPtr) and the number of bytes to be copied (SduLength). If not enough transmit data is available, no data is copied by the upper layer module and BUFREQ_E_BUSY is returned. The lower layer module may retry the call. An SduLength of 0 can be used to indicate state changes in the retry parameter or to query the current amount of available data in the upper layer module. In this case, the Sdu DataPtr may be a NULL_PTR.
	retry	This parameter is used to acknowledge transmitted data or to retransmit data after transmission problems. If the retry parameter is a NULL_PTR, it indicates that the transmit data can be removed from the buffer immediately after it has been copied. Otherwise, the retry parameter must point to a valid RetryInfoType element. If TpDataState indicates TP_CONFPENDING, the previously copied data must remain in the TP buffer to be available for error recovery. TP_DATACONF indicates that all data that has been copied before this call is confirmed and can be removed from the TP buffer. Data copied by this API call is excluded and will be confirmed later. TP_DATARETRY indicates that this API call shall copy previously copied data in order to recover from an error. In this case TxTpDataCnt specifies the offset in bytes from the current data copy position.
Parameters (inout)	None	

▽

△

Parameters (out)	availableDataPtr	Indicates the remaining number of bytes that are available in the upper layer module's Tx buffer. availableDataPtr can be used by TP modules that support dynamic payload lengths (e.g. FrIsoTp) to determine the size of the following CFs.
Return value	BufReq_ReturnType	`BUFREQ_OK`: Data has been copied to the transmit buffer completely as requested. `BUFREQ_E_BUSY`: Request could not be fulfilled, because the required amount of Tx data is not available. The lower layer module may retry this call later on. No data has been copied. `BUFREQ_E_NOT_OK`: Data has not been copied. Request failed.
Description	This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the I-PDU data unless retry->TpDataState is TP_DATARETRY. In this case the function restarts to copy the data beginning at the offset from the current position indicated by retry->TxTpDataCnt. The size of the remaining data is written to the position indicated by availableDataPtr.	
Available via	VdpCmR_Externals.h	

⌋

8.4.1.5 VdpCmR_TpTxConfirmation

[CP_SWS_VdpCmR_01304] Definition of callback function VdpCmR_TpTxConfirmation

Upstream requirements: FO_RS_VDP_00004, FO_RS_VDP_00010

⌈

Service Name	VdpCmR_TpTxConfirmation	
Syntax	`void VdpCmR_TpTxConfirmation ( PduIdType id, Std_ReturnType result )`	
Service ID [hex]	0x48	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the transmitted I-PDU.
	result	E_OK: The PDU was transmitted. E_NOT_OK: Transmission of the PDU failed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This function is called after the I-PDU has been transmitted on its network, the result indicates whether the transmission was successful or not.	
Available via	VdpCmR_Externals.h	

⌋

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1 VdpCmR_MainFunction

[CP_SWS_VdpCmR_01400] Definition of scheduled function VdpCmR_MainFunction

Upstream requirements: SRS_BSW_00373

⌈

Service Name	VdpCmR_MainFunction
Syntax	`void VdpCmR_MainFunction ( void )`
Service ID [hex]	0x22
Description	This cyclic main function handles incoming requests, outgoing responses and data messages.
Available via	VdpCmR.h

⌋

8.6 Expected interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory interfaces

This section defines all interfaces, which are required to fulfill the core functionality of the module.

[CP_SWS_VdpCmR_01500] Definition of mandatory interfaces required by module VdpCmR

Upstream requirements: SRS_BSW_00384

⌈

API Function	***Header File***	***Description***
PduR_VdpCmRTransmit	PduR_VdpCmR.h	Requests transmission of a PDU.
StbM_GetCurrentTime	StbM.h	Returns a time tuple (Local time, Global time and Timebase status) and user data details Note: This API shall be called with locked interrupts / within an Exclusive Area to prevent interruption (i.e., the risk that the time stamp is outdated on return of the function call).

⌋

8.6.2 Optional interfaces

This section defines all interfaces, which are required to fulfill an optional functionality of the module.

[CP_SWS_VdpCmR_01501] Definition of optional interfaces requested by module VdpCmR

Upstream requirements: SRS_BSW_00384

⌈

API Function	*Header File*	*Description*
Det_ReportError	Det.h	Service to report development errors.
NvM_GetErrorStatus	NvM.h	Service to read the block dependent error/status information.
NvM_ReadBlock	NvM.h	Service to copy the data of the NV block to its corresponding RAM block.
NvM_SetRamBlockStatus	NvM.h	Service for setting the RAM block status of a permanent RAM block or the status of the explicit synchronization of a NVRAM block.
NvM_WriteBlock	NvM.h	Service to copy the data of the RAM block to its corresponding NV block.

⌋

8.6.3 Configurable interfaces

In this section, all interfaces are listed where the target function could be configured. The target function is usually a callback function. The names of this kind of interfaces are not fixed because they are configurable.

The interface from `VdpCmR` to the `DCA Remotes` is configurable. The API is specified by the `VdpCmR`, the implementation needs to be realized inside the specific `DCA Remote`.

<DcaRemote> shall be replaced by the configured name of the enclosing container *VdpCmRDcaRemote*.

8.6.3.1 <DcaRemote>_AddDpDCfg

[CP_SWS_VdpCmR_01600] Definition of configurable interface <DcaRemote>_AddDpDCfg

Upstream requirements: FO_RS_VDP_00002

⌈

Service Name	<DcaRemote>_AddDpDCfg	
Syntax	`Std_ReturnType <DcaRemote>_AddDpDCfg ( const VdpCmR_Dca_VdpSlotIdDpDCfgPairType* ptrVdpSlotIdDCfgPair )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ptrVdpSlotIdDCfgPair	Contains the VDP Slot ID and its respective data point configuration.

▽

△

Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK, E_NOT_OK, VDPCMR_DCA_E_VDP_SLOT_ID_ALREADY_IN_USE, VDPCMR_DCA_E_INVALID_DCA_REM_DP_DCFG, VDPCMR_DCA_E_DUPLICATE_DCA_REM_DP_DCFG, VDPCMR_DCA_E_DP_LIMIT_REACHED, VDPCMR_DCA_E_SAMPLING_MODE_CYCLIC_NOT_SUPPORTED, VDPCMR_DCA_E_SAMPLING_MODE_ON_CHANGE_NOT_SUPPORTED, VDPCMR_DCA_E_SAMPLING_MODE_CYCLIC_AND_ON_CHANGE_NOT_SUPPORTED, VDPCMR_DCA_E_CUSTOM_ERROR_0 .. VDPCMR_DCA_E_CUSTOM_ERROR_47 allowed
Description	The VdpCmR uses this function to add a new dynamic configuration to the DCA Remote.	
Available via	<DcaRemote>.h	

⌋

8.6.3.2 <DcaRemote>_SetDpActivationState

[CP_SWS_VdpCmR_01601] Definition of configurable interface <DcaRemote>_SetDpActivationState

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003

⌈

Service Name	<DcaRemote>_SetDpActivationState	

Syntax	`Std_ReturnType <DcaRemote>_SetDpActivationState (`
	`  uint16 vdpSlotId,`
	`  VdpCmR_Dca_DpActiveStateEnumType activeState`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	vdpSlotId	The VDP Slot ID of the data point which shall be activated or deactivated.
	activeState	Indicates whether sampling should start or stop
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK, E_NOT_OK, VDPCMR_DCA_E_UNKNOWN_VDP_SLOT_ID, VDPCMR_DCA_E_CUSTOM_ERROR_0 .. VDPCMR_DCA_E_CUSTOM_ERROR_47 allowed.
Description	The VdpCmR uses this function to activate or deactivate a data point inside the DCA Remote.	
Available via	<DcaRemote>.h	

⌋

8.6.3.3 <DcaRemote>_TriggerDpOneTimeSampling

[CP_SWS_VdpCmR_01602] Definition of configurable interface <DcaRemote>_TriggerDpOneTimeSampling

Upstream requirements: FO_RS_VDP_00002, FO_RS_VDP_00003

⌈

Service Name	<DcaRemote>_TriggerDpOneTimeSampling	
Syntax	`Std_ReturnType <DcaRemote>_TriggerDpOneTimeSampling ( uint16 vdpSlotId )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	vdpSlotId	The VDP Slot ID of the data point which shall be sampled once.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK, E_NOT_OK, VDPCMR_DCA_E_UNKNOWN_VDP_SLOT_ID, VDPCMR_DCA_E_SAMPLING_ERROR allowed.
Description	The VdpCmR uses this function to trigger one time sampling of the data point inside the DCA Remote.	
Available via	<DcaRemote>.h	

⌋

8.6.3.4 <DcaRemote>_DeleteDpDCfg

[CP_SWS_VdpCmR_01603] Definition of configurable interface <DcaRemote>_DeleteDpDCfg

Upstream requirements: FO_RS_VDP_00002

⌈

Service Name	<DcaRemote>_DeleteDpDCfg	
Syntax	`void <DcaRemote>_DeleteDpDCfg ( uint16 vdpSlotId )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	vdpSlotId	The VDP Slot ID of the data point which shall be removed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	The VdpCmR uses this function to remove one data point inside the DCA Remote.	
Available via	<DcaRemote>.h	

⌋

8.6.3.5 <DcaRemote>_DeleteAllDpDCfgs

[CP_SWS_VdpCmR_01604] Definition of configurable interface <DcaRemote>_DeleteAllDpDCfgs

Upstream requirements: FO_RS_VDP_00002

⌈

Service Name	<DcaRemote>_DeleteAllDpDCfgs

Syntax	`void <DcaRemote>_DeleteAllDpDCfgs (`
	`  void`
	`)`

Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	The VdpCmR uses this function to remove all data points from the DCA Remote.
Available via	<DcaRemote>.h

⌋

8.7 Service Interfaces

There are no service interfaces.

9 Sequence diagrams

The following sequence diagram shows a communication example sequence, where an <DcaRemote>_AddDpDCfg is passed from the PduR to the VdpCmR and from there to the DCA Remotes.

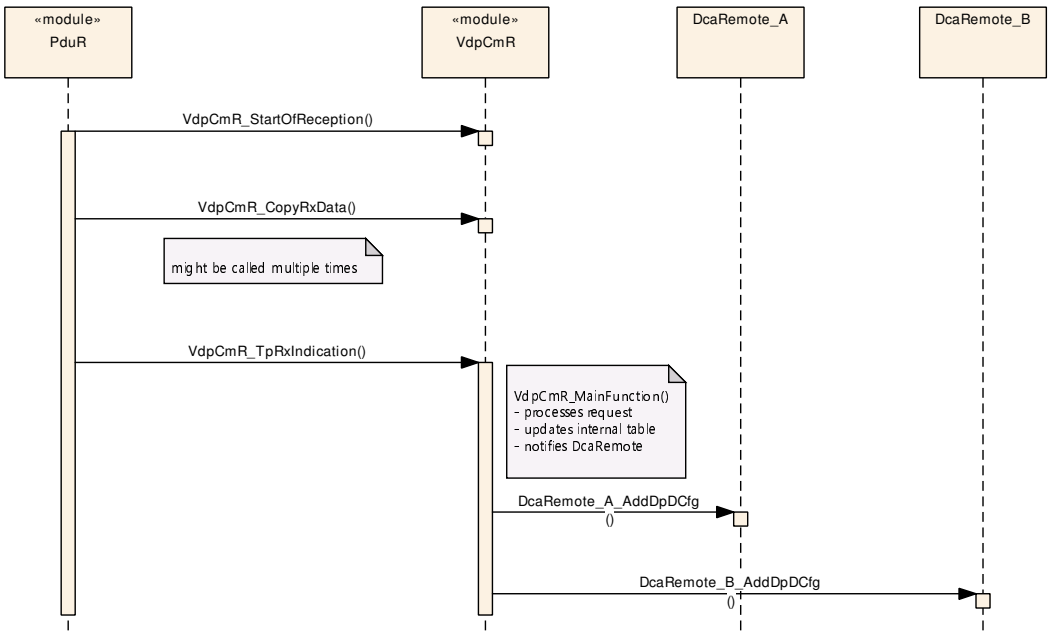

Figure 9.1: Communication example from PduR through VdpCmR to DCA Remotes.

The following sequence diagram shows a communication example sequence, where a data sample is passed through VdpCmR_AddSample from different DCA Remotes to the VdpCmR. Later in the call to VdpCmR_MainFunction it passes the data message to the PduR.

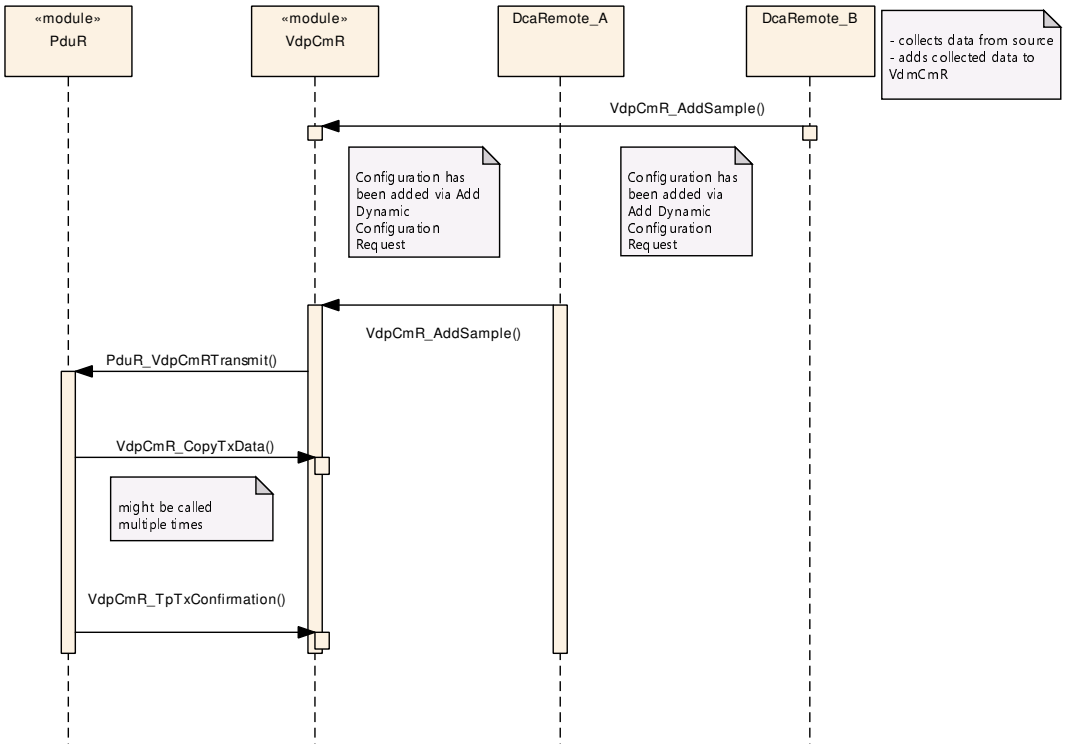

Figure 9.2: Communication example from DCA Remotes through VdpCmR to PduR.

10 Configuration specification

This chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals.

Chapter 10.2 specifies the structure (containers) and the parameters of the module VdpCmR.

Chapter 10.3 specifies published information of the module VdpCmR.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in [3].

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

10.2.1 VdpCmR

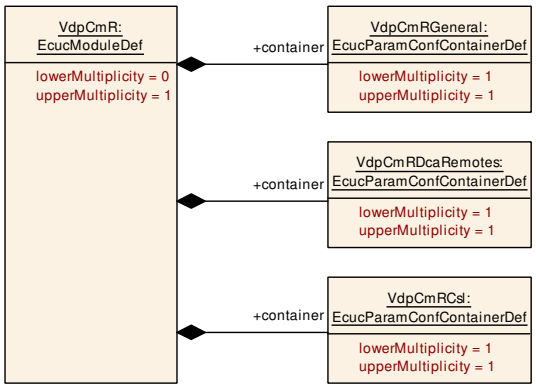

Figure 10.1: VdpCmR

[ECUC_VdpCmR_00001] Definition of EcucModuleDef VdpCmR ⌈

Module Name	VdpCmR
Description	The Vehicle Data Protocol is a protocol to sample data on ECUs with low computational overhead and memory usage. It is configured using this module.
Post-Build Variant Support	false
Supported Config Variants	VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Dependency**
VdpCmRCsl	1	This container specifies the communication and security layer.
VdpCmRDcaRemotes	1	This container specifies every DCA Remote on the ECU that shall be connected to the VdpCmR via a pre-compile time configuration path
VdpCmRGeneral	1	This container specifies general parameters for VdpCmR.

⌋

10.2.2 VdpCmRGeneral

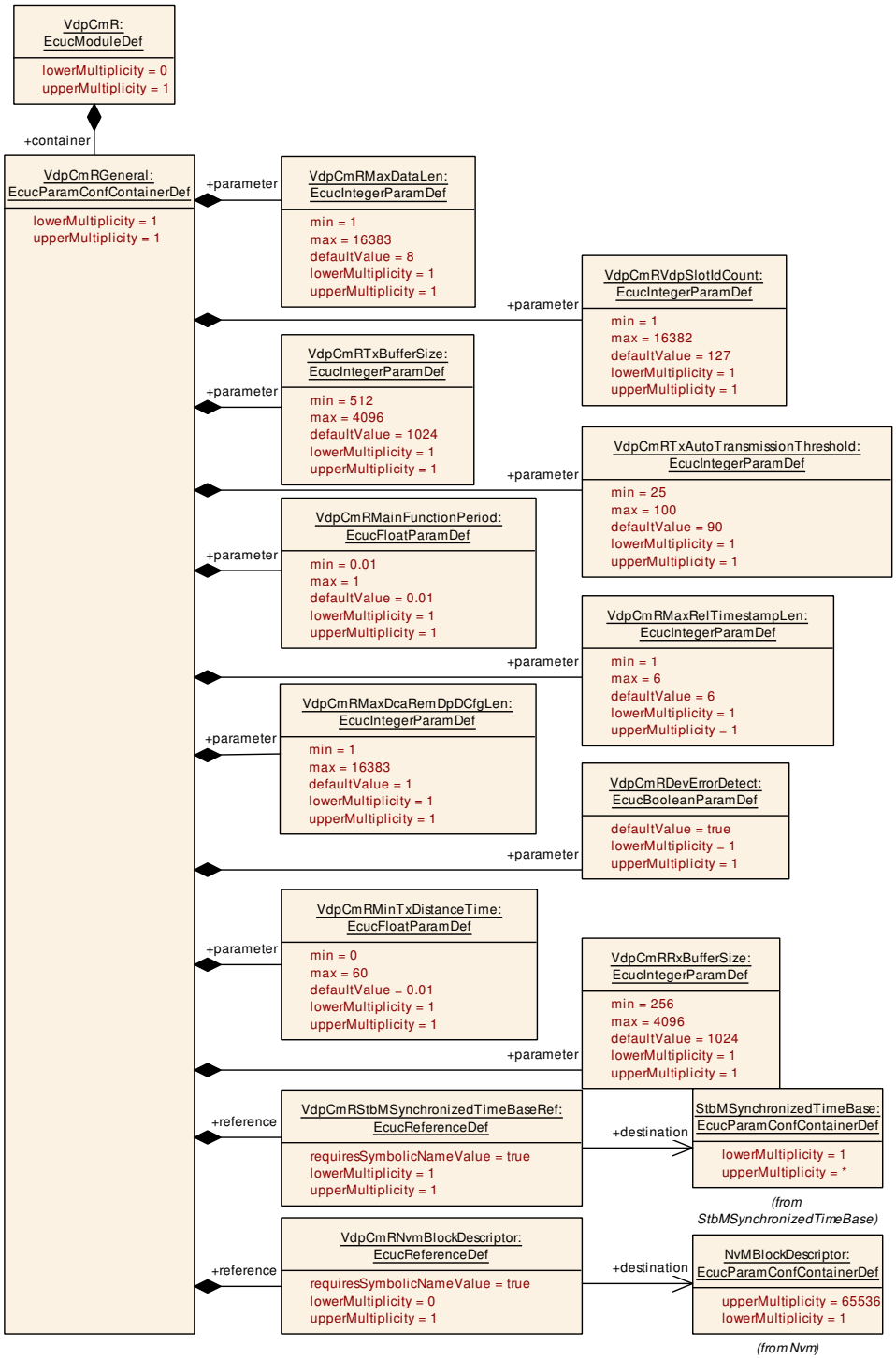

Figure 10.2: VdpCmRGeneral

[ECUC_VdpCmR_00002] Definition of EcucParamConfContainerDef VdpCmRGeneral ⌈

Container Name	VdpCmRGeneral
Parent Container	VdpCmR
Description	This container specifies general parameters for VdpCmR.
Multiplicity	1
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
VdpCmRDevErrorDetect	1	[ECUC_VdpCmR_00012]
VdpCmRMainFunctionPeriod	1	[ECUC_VdpCmR_00009]
VdpCmRMaxDataLen	1	[ECUC_VdpCmR_00005]
VdpCmRMaxDcaRemDpDCfgLen	1	[ECUC_VdpCmR_00011]
VdpCmRMaxRelTimestampLen	1	[ECUC_VdpCmR_00010]
VdpCmRMinTxDistanceTime	1	[ECUC_VdpCmR_00013]
VdpCmRRxBufferSize	1	[ECUC_VdpCmR_00014]
VdpCmRTxAutoTransmissionThreshold	1	[ECUC_VdpCmR_00008]
VdpCmRTxBufferSize	1	[ECUC_VdpCmR_00007]
VdpCmRVdpSlotIdCount	1	[ECUC_VdpCmR_00006]
VdpCmRNvmBlockDescriptor	0..1	[ECUC_VdpCmR_00016]
VdpCmRStbMSynchronizedTimeBaseRef	1	[ECUC_VdpCmR_00015]

No Included Containers

⌋

[ECUC_VdpCmR_00012] Definition of EcucBooleanParamDef VdpCmRDevErrorDetect ⌈

Parameter Name	VdpCmRDevErrorDetect		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies whether or not the development error detection is enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	true		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00009] Definition of EcucFloatParamDef VdpCmRMainFunctionPeriod ⌈

Parameter Name	VdpCmRMainFunctionPeriod		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies the main function period in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0.01 .. 1]		
Default value	0.01		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00005] Definition of EcucIntegerParamDef VdpCmRMaxDataLen ⌈

Parameter Name	VdpCmRMaxDataLen		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies the maximum length in bytes of the data field in a data message.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 16383		
Default value	8		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00011] Definition of EcucIntegerParamDef VdpCmRMaxDcaRemDpDCfgLen ⌈

Parameter Name	VdpCmRMaxDcaRemDpDCfgLen		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies the maximum length in bytes of the DCA Remote Data Point Configuration field. Note: This parameter has an impact on RAM usage.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 16383		

▽

△

Default value	1		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00010] Definition of EcucIntegerParamDef VdpCmRMaxRelTimestampLen ⌈

Parameter Name	VdpCmRMaxRelTimestampLen		
Parent Container	VdpCmRGeneral		
Description	This parameter defines the maximum encodable length in bytes of a relative timestamp.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 6		
Default value	6		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00013] Definition of EcucFloatParamDef VdpCmRMinTxDistanceTime ⌈

Parameter Name	VdpCmRMinTxDistanceTime		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies the minimum transmission distance time, i.e. the minimum time that has to pass between two consecutive VDP data messages. Note: This parameter directly limits the bus load that is created by the VdpCmR. If configured too high, it might induce data loss. Unit:seconds		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. 60]		
Default value	0.01		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00014] Definition of EcucIntegerParamDef VdpCmRRxBuffer Size ⌈

Parameter Name	VdpCmRRxBufferSize		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies the size in bytes of the reception buffer.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	256 .. 4096		
Default value	1024		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00008] Definition of EcucIntegerParamDef VdpCmRTxAuto TransmissionThreshold ⌈

Parameter Name	VdpCmRTxAutoTransmissionThreshold		
Parent Container	VdpCmRGeneral		
Description	This parameter defines the threshold (in percentage) for the data message buffer at which an automatic transmission is requested. This means whenever this threshold is exceeded, the component will initiate a transmission of the buffer as soon as possible. This parameter can prevent data loss, depending on the setup. Note: It is recommended to set this parameter based on the expected throughput of the DCA Remotes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	25 .. 100		
Default value	90		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00007] Definition of EcucIntegerParamDef VdpCmRTxBuffer Size ⌈

Parameter Name	VdpCmRTxBufferSize
Parent Container	VdpCmRGeneral
Description	Description: This parameter specifies the buffer size in bytes of the transmission buffers..
Multiplicity	1

▽

△

Type	EcucIntegerParamDef		
Range	512 .. 4096		
Default value	1024		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00006] Definition of EcucIntegerParamDef VdpCmRVdpSlotId Count ⌈

Parameter Name	VdpCmRVdpSlotIdCount		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies the maximum number of VDP Slot IDs that can be configured simultaneously. Note: The VDP Slot ID 16383 is always reserved for error handling.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 16382		
Default value	127		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00016] Definition of EcucReferenceDef VdpCmRNvmBlockDescriptor ⌈

Parameter Name	VdpCmRNvmBlockDescriptor		
Parent Container	VdpCmRGeneral		
Description	This reference is used to load and store the non-volatile data of the VdpCmR module. The supported NvM block name is: VdpCmR_NvmRomBlockData		
Multiplicity	0..1		
Type	Symbolic name reference to NvMBlockDescriptor		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00015] Definition of EcucReferenceDef VdpCmRStbMSynchronizedTimeBaseRef ⌈

Parameter Name	VdpCmRStbMSynchronizedTimeBaseRef		
Parent Container	VdpCmRGeneral		
Description	This parameter specifies the StbM synchronized time base reference of the VdpCmR.		
Multiplicity	1		
Type	Symbolic name reference to StbMSynchronizedTimeBase		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

10.2.3 VdpCmRDcaRemotes

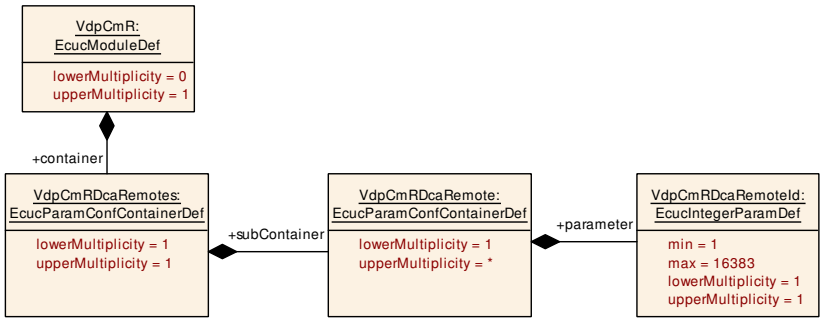

Figure 10.3: VDP-CM Remote DCA

[ECUC_VdpCmR_00003] Definition of EcucParamConfContainerDef VdpCmRDcaRemotes ⌈

Container Name	VdpCmRDcaRemotes
Parent Container	VdpCmR
Description	This container specifies every DCA Remote on the ECU that shall be connected to the VdpCmR via a pre-compile time configuration path
Multiplicity	1
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Dependency**
VdpCmRDcaRemote	1..*	This container specifies the general parameters of a DCA Remote entity. Please note that the actual multiplicity of this container also defines the number of DCA remotes. The shortName of the container shall be used to disambiguate the configurable interfaces of the DCA remotes.

⌋

[ECUC_VdpCmR_00018] Definition of EcucParamConfContainerDef VdpCmRDcaRemote ⌈

Container Name	VdpCmRDcaRemote
Parent Container	VdpCmRDcaRemotes
Description	This container specifies the general parameters of a DCA Remote entity. Please note that the actual multiplicity of this container also defines the number of DCA remotes. The shortName of the container shall be used to disambiguate the configurable interfaces of the DCA remotes.
Multiplicity	1..*
Post-Build Variant Multiplicity	false
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
VdpCmRDcaRemoteId	1	[ECUC_VdpCmR_00019]

No Included Containers

⌋

[ECUC_VdpCmR_00019] Definition of EcucIntegerParamDef VdpCmRDcaRemoteId ⌈

Parameter Name	VdpCmRDcaRemoteId		
Parent Container	VdpCmRDcaRemote		
Description	This parameter specifies the DCA Remote ID		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 16383		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

10.2.4 VdpCmRCsl

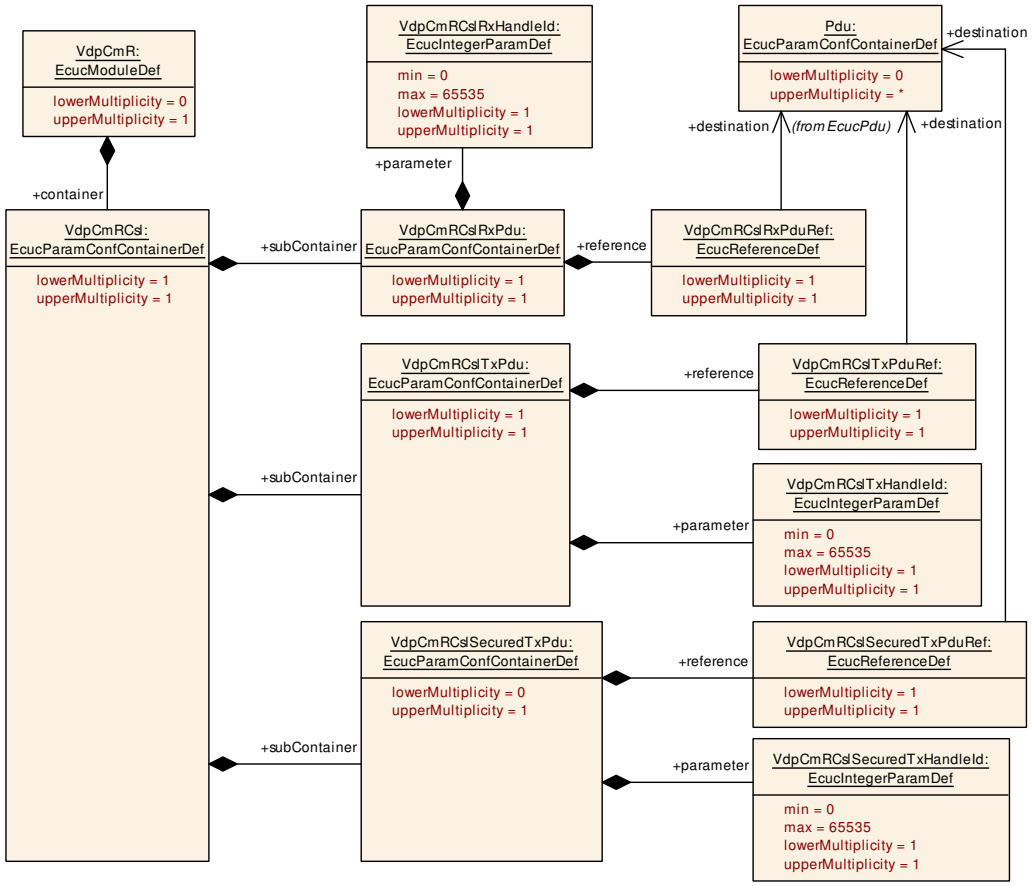

Figure 10.4: VDP-CM Remote Csl

[SWS_VdpCmR_CONSTR_00001] Restriction for `VdpCmRCslTxPdu.VdpCmRCslTxPduRef` ⌈The only valid `destination` of `VdpCmRCslTxPdu.VdpCmRCslTxPduRef` is a Pdu that is derived from a `GeneralPurposeIPdu` where attribute `category` is set to the value `VDP`.⌋

[SWS_VdpCmR_CONSTR_00002] Restriction for `VdpCmRCslRxPdu.VdpCmRCslRxPduRef` ⌈The only valid `destination` of `VdpCmRCslRxPdu.VdpCmRCslRxPduRef` is a Pdu that is derived from a `GeneralPurposeIPdu` where attribute `category` is set to the value `VDP`.⌋

[ECUC_VdpCmR_00004] Definition of EcucParamConfContainerDef VdpCmRCsl ⌈

Container Name	VdpCmRCsl
Parent Container	VdpCmR
Description	This container specifies the communication and security layer.
Multiplicity	1
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Dependency**
VdpCmRCslRxPdu	1	This container specifies the general rx message.
VdpCmRCslSecuredTxPdu	0..1	This container specifies the secured tx message.
VdpCmRCslTxPdu	1	This container specifies the general tx message.

⌋

[ECUC_VdpCmR_00025] Definition of EcucParamConfContainerDef VdpCmRCslTxPdu ⌈

Container Name	VdpCmRCslTxPdu
Parent Container	VdpCmRCsl
Description	This container specifies the general tx message.
Multiplicity	1
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
VdpCmRCslTxHandleId	1	[ECUC_VdpCmR_00026]
VdpCmRCslTxPduRef	1	[ECUC_VdpCmR_00027]

No Included Containers

⌋

[ECUC_VdpCmR_00026] Definition of EcucIntegerParamDef VdpCmRCslTxHandleId ⌈

Parameter Name	VdpCmRCslTxHandleId		
Parent Container	VdpCmRCslTxPdu		
Description	Symbolic handle for the Pdu, needs to be unique in the context of VdpCmR.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00027] Definition of EcucReferenceDef VdpCmRCslTxPduRef ⌈

Parameter Name	VdpCmRCslTxPduRef		
Parent Container	VdpCmRCslTxPdu		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00022] Definition of EcucParamConfContainerDef VdpCmRCslRxPdu ⌈

Container Name	VdpCmRCslRxPdu
Parent Container	VdpCmRCsl
Description	This container specifies the general rx message.
Multiplicity	1
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
VdpCmRCslRxHandleId	1	[ECUC_VdpCmR_00023]
VdpCmRCslRxPduRef	1	[ECUC_VdpCmR_00024]

No Included Containers

⌋

[ECUC_VdpCmR_00023] Definition of EcucIntegerParamDef VdpCmRCslRxHandleId ⌈

Parameter Name	VdpCmRCslRxHandleId		
Parent Container	VdpCmRCslRxPdu		
Description	Symbolic handle for the Pdu, needs to be unique in the context of VdpCmR.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants

▽

△

	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00024] Definition of EcucReferenceDef VdpCmRCslRxPduRef ⌈

Parameter Name	VdpCmRCslRxPduRef		
Parent Container	VdpCmRCslRxPdu		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00028] Definition of EcucParamConfContainerDef VdpCmRCslSecuredTxPdu ⌈

Container Name	VdpCmRCslSecuredTxPdu
Parent Container	VdpCmRCsl
Description	This container specifies the secured tx message.
Multiplicity	0..1
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
VdpCmRCslSecuredTxHandleId	1	[ECUC_VdpCmR_00029]
VdpCmRCslSecuredTxPduRef	1	[ECUC_VdpCmR_00030]

No Included Containers

⌋

[ECUC_VdpCmR_00029] Definition of EcucIntegerParamDef VdpCmRCslSecuredTxHandleId ⌈

Parameter Name	VdpCmRCslSecuredTxHandleId
Parent Container	VdpCmRCslSecuredTxPdu
Description	Symbolic handle for the Pdu, needs to be unique in the context of VdpCmR.

▽

△

Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

[ECUC_VdpCmR_00030] Definition of EcucReferenceDef VdpCmRCslSecuredTxPduRef ⌈

Parameter Name	VdpCmRCslSecuredTxPduRef		
Parent Container	VdpCmRCslSecuredTxPdu		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Dependency			

⌋

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in [3].

A Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

This chapter is generated.

Class	**EcucReferenceDef**			
Note	Specify references within the ECU Configuration Description between parameter containers. This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARObject*, *AtpDefinition*, *EcucAbstractInternalReferenceDef*, *EcucAbstractReferenceDef*, *EcucCommonAttributes*, *EcucDefinitionElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	EcucDestinationUriPolicy.reference, EcucParamConfContainerDef.reference			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
destination	EcucContainerDef	0..1	ref	Exactly one reference to a parameter container is allowed as destination. **Stereotypes:** atpUriDef

Table A.1: EcucReferenceDef

Class	**GeneralPurposeIPdu**			
Note	This element is used for AUTOSAR Pdus without attributes that are routed by the PduR. Please note that the category name of such Pdus is standardized in the AUTOSAR System Template. **Tags:** atp.recommendedPackage=Pdus			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *IPdu*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Pdu*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.2: GeneralPurposeIPdu

Class	***Identifiable*** (abstract)
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, AppOsTask ProxyToEcuTaskProxyMapping, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartition Mapping, AppliedStandard, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *Atp Classifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BinaryManifest AddressableObject*, BinaryManifestItemDefinition, *BinaryManifestResource*, BinaryManifestResource Definition, BlockState, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, Build ActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClientIdDefinition, Client ServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, *CouplingElementAbstractDetails*, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPort StructuralElement*, *CpSoftwareClusterResource*, CpSoftwareClusterResourceToApplicationPartition Mapping, CpSoftwareClusterToApplicationPartitionMapping, CpSoftwareClusterToEcuInstanceMapping, CpSoftwareClusterToResourceMapping, *CryptoServiceMapping*, CyclicHandlingComDataToOsTask ProxyMapping, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, DataTransformation, *Dds AbstractServiceInstanceElementCp*, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, ▽

▽

△

Class	**Identifiable** (abstract)			
	△ DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticExtendedDataRecordElement, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltArgumentProps, DltLogChannel, DltMessage, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, ECUMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcuPartitionToCoreMapping, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, EthernetWakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722TpAcfBusPart*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, J1939Node, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacAddressVlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, ModeSwitchSenderComSpecProps, NetworkEndpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PhysicalChannel*, PortElementToCommunicationResourceMapping, PortGroup, *PortInterfaceMapping*, QueuedReceiverComSpecProps, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RteEventInCompositionSeparation, RteEventInCompositionToOsTaskProxyMapping, RteEventInSystemSeparation, RteEventInSystemToOsTaskProxyMapping, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobRequirement, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServerCallPoint*, ServerComSpecProps, *ServiceNeeds*, SignalServiceTranslationElementProps, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *StackUsage*, StaticSocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SwitchAsynchronousTrafficShaperGroupEntry, SwitchAtsInstanceEntry, SwitchFlowMeteringEntry, SwitchStreamFilterActionDestPortModification, SwitchStreamFilterEntry, SwitchStreamFilterRule, SwitchStreamGateEntry, SwitchStreamIdentification, SystemMapping, SystemSignalGroupToCommunicationResourceMapping, SystemSignalToCommunicationResourceMapping, TDCpSoftwareClusterMapping, TDCpSoftwareClusterResourceMapping, TcpOptionFilterList, *TimingClock*, TimingClockSyncAccuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:** xml.sequenceOffset=-50

▽

△

Class	***Identifiable*** (abstract)			
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:** xml.attribute=true

Table A.3: Identifiable

B Not applicable requirements

None.

C Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

C.1 Traceable item history of this document according to AUTOSAR Release R25-11

C.1.1 Added Specification Items in R25-11

Number	Heading
[CP_SWS_VdpCmR_00100]	Initialization
[CP_SWS_VdpCmR_00101]	Incoming Messages
[CP_SWS_VdpCmR_00102]	Receive buffer
[CP_SWS_VdpCmR_00103]	Receive StartOfReception
[CP_SWS_VdpCmR_00104]	Receive CopyRxData
[CP_SWS_VdpCmR_00105]	Receive TpRxIndication
[CP_SWS_VdpCmR_00106]	Receive buffer processing
[CP_SWS_VdpCmR_00107]	Incoming VDP Requests
[CP_SWS_VdpCmR_00108]	Get protocol version
[CP_SWS_VdpCmR_00109]	Control message types
[CP_SWS_VdpCmR_00110]	Add Dynamic Configuration - only cyclic transmission
[CP_SWS_VdpCmR_00111]	Add Dynamic Configuration - DCA remote dynamic configuration
[CP_SWS_VdpCmR_00112]	Add Dynamic Configuration - Construction of VdpCmR_Dca_VdpSlotIdDpDCfgPairType
[CP_SWS_VdpCmR_00113]	Add Dynamic Configuration - Sampling Mode
[CP_SWS_VdpCmR_00114]	Add Dynamic Configuration - Sampling Mode
[CP_SWS_VdpCmR_00115]	Add Dynamic Configuration - VDP Settings
[CP_SWS_VdpCmR_00116]	Add Dynamic Configuration - Returned errors
[CP_SWS_VdpCmR_00119]	Maximum VDP Slot ID
[CP_SWS_VdpCmR_00120]	Remove Dynamic Configuration Request
[CP_SWS_VdpCmR_00121]	Remove Dynamic Configuration Request - cyclic
[CP_SWS_VdpCmR_00122]	Remove Dynamic Configuration Request - VDP Slot ID
[CP_SWS_VdpCmR_00123]	Remove Dynamic Configuration Request - Global
[CP_SWS_VdpCmR_00124]	Remove Dynamic Configuration Request - DCA
[CP_SWS_VdpCmR_00130]	Activation state
[CP_SWS_VdpCmR_00131]	Activation request payload
[CP_SWS_VdpCmR_00132]	Activation at remote DCA

△

Number	Heading
[CP_SWS_VdpCmR_00133]	Activation request - Validation in VdpCmR
[CP_SWS_VdpCmR_00134]	Activation request - Sampling Error in DCA Remote
[CP_SWS_VdpCmR_00135]	Activation request - Response
[CP_SWS_VdpCmR_00140]	Trigger request - Transmission
[CP_SWS_VdpCmR_00141]	Trigger request - Sampling
[CP_SWS_VdpCmR_00142]	Trigger request - Sampling - Validation in VdpCmR
[CP_SWS_VdpCmR_00143]	Trigger request - Sampling Error in DCA Remote
[CP_SWS_VdpCmR_00144]	Trigger request - Response
[CP_SWS_VdpCmR_00150]	Communication configuration
[CP_SWS_VdpCmR_00151]	Unsecure and secure PDU reception
[CP_SWS_VdpCmR_00152]	Rx PDU configuration
[CP_SWS_VdpCmR_00153]	Unsecure and secure PDU transmission
[CP_SWS_VdpCmR_00154]	Unsecured Tx PDU configuration
[CP_SWS_VdpCmR_00155]	Secured Tx PDU configuration
[CP_SWS_VdpCmR_00210]	DCA Remote count
[CP_SWS_VdpCmR_00211]	DCA Remote names
[CP_SWS_VdpCmR_00212]	DCA Remote ID mapping
[CP_SWS_VdpCmR_00213]	DCA Remote ID unknown
[CP_SWS_VdpCmR_00401]	Adding data point samples
[CP_SWS_VdpCmR_00402]	Adding too big sample
[CP_SWS_VdpCmR_00403]	Consecutive data sample in message
[CP_SWS_VdpCmR_00404]	Data
[CP_SWS_VdpCmR_00405]	On-sampling transmission
[CP_SWS_VdpCmR_00406]	First data sample in message
[CP_SWS_VdpCmR_00407]	AddSample Buffer full
[CP_SWS_VdpCmR_00408]	Maximum timestamp length
[CP_SWS_VdpCmR_00409]	Timebase error
[CP_SWS_VdpCmR_00410]	Get current timestamp
[CP_SWS_VdpCmR_00501]	Adding asynchronous errors
[CP_SWS_VdpCmR_00503]	Error Information
[CP_SWS_VdpCmR_00504]	Asynchronous Error already contained
[CP_SWS_VdpCmR_00505]	Data Source Error
[CP_SWS_VdpCmR_00506]	AddAsyncError Buffer full
[CP_SWS_VdpCmR_00601]	Persistency
[CP_SWS_VdpCmR_00602]	NvM Block Descriptor
[CP_SWS_VdpCmR_00603]	NvM Block Name
[CP_SWS_VdpCmR_00604]	Read Persistent Data

▽

△

Number	Heading
[CP_SWS_VdpCmR_00605]	Write Persistent Data
[CP_SWS_VdpCmR_00606]	Get Persistent Data Error Status
[CP_SWS_VdpCmR_00607]	NvM RAM Block Mirror
[CP_SWS_VdpCmR_00608]	Enable Persistence
[CP_SWS_VdpCmR_00609]	Load Persistent Configuration
[CP_SWS_VdpCmR_00610]	Load Persistent Configuration Fails
[CP_SWS_VdpCmR_00611]	Store Persistent Configuration
[CP_SWS_VdpCmR_00701]	Cyclic call VdpCmR_MainFunction
[CP_SWS_VdpCmR_00702]	Cyclic transmission
[CP_SWS_VdpCmR_00703]	Buffer threshold exceeded
[CP_SWS_VdpCmR_00704]	Transmission and MTDT
[CP_SWS_VdpCmR_00705]	Data Message Buffer Locked
[CP_SWS_VdpCmR_00706]	Transmission copy data
[CP_SWS_VdpCmR_00707]	Transmission confirmation
[CP_SWS_VdpCmR_01000]	Definition of development errors in module VdpCmR
[CP_SWS_VdpCmR_01001]	VdpCmR Error Not Initialized
[CP_SWS_VdpCmR_01002]	VdpCmR Error Invalid Argument
[CP_SWS_VdpCmR_01003]	VdpCmR Error Data Length Exceeded
[CP_SWS_VdpCmR_01100]	Definition of Std_ReturnType-extension for module VdpCmR
[CP_SWS_VdpCmR_01101]	Definition of datatype VdpCmR_ConfigType
[CP_SWS_VdpCmR_01102]	Definition of datatype VdpCmR_Dca_DpActiveStateEnumType
[CP_SWS_VdpCmR_01103]	Definition of datatype VdpCmR_Dca_SamplingModeEnumType
[CP_SWS_VdpCmR_01104]	Definition of datatype VdpCmR_Dca_VdpSlotIdDpDCfgPairType
[CP_SWS_VdpCmR_01200]	Definition of API function VdpCmR_Init
[CP_SWS_VdpCmR_01201]	VdpCmR ConfigPtr at Init
[CP_SWS_VdpCmR_01202]	Definition of API function VdpCmR_GetVersionInfo
[CP_SWS_VdpCmR_01203]	Definition of API function VdpCmR_AddSample
[CP_SWS_VdpCmR_01204]	Definition of API function VdpCmR_AddAsyncError
[CP_SWS_VdpCmR_01300]	Definition of callback function VdpCmR_StartOfReception
[CP_SWS_VdpCmR_01301]	Definition of callback function VdpCmR_CopyRxData
[CP_SWS_VdpCmR_01302]	Definition of callback function VdpCmR_TpRxIndication
[CP_SWS_VdpCmR_01303]	Definition of callback function VdpCmR_CopyTxData
[CP_SWS_VdpCmR_01304]	Definition of callback function VdpCmR_TpTxConfirmation
[CP_SWS_VdpCmR_01400]	Definition of scheduled function VdpCmR_MainFunction
[CP_SWS_VdpCmR_01500]	Definition of mandatory interfaces required by module VdpCmR
[CP_SWS_VdpCmR_01501]	Definition of optional interfaces requested by module VdpCmR
[CP_SWS_VdpCmR_01600]	Definition of configurable interface <DcaRemote>_AddDpDCfg

▽

△

Number	Heading
[CP_SWS_VdpCmR_01601]	Definition of configurable interface <DcaRemote>_SetDpActivation State
[CP_SWS_VdpCmR_01602]	Definition of configurable interface <DcaRemote>_TriggerDpOne TimeSampling
[CP_SWS_VdpCmR_01603]	Definition of configurable interface <DcaRemote>_DeleteDpDCfg
[CP_SWS_VdpCmR_01604]	Definition of configurable interface <DcaRemote>_DeleteAllDp DCfgs
[CP_SWS_VdpCmR_02000]	Definition of imported datatypes of module VdpCmR
[ECUC_VdpCmR_00001]	Definition of EcucModuleDef VdpCmR
[ECUC_VdpCmR_00002]	Definition of EcucParamConfContainerDef VdpCmRGeneral
[ECUC_VdpCmR_00003]	Definition of EcucParamConfContainerDef VdpCmRDcaRemotes
[ECUC_VdpCmR_00004]	Definition of EcucParamConfContainerDef VdpCmRCsl
[ECUC_VdpCmR_00005]	Definition of EcucIntegerParamDef VdpCmRMaxDataLen
[ECUC_VdpCmR_00006]	Definition of EcucIntegerParamDef VdpCmRVdpSlotIdCount
[ECUC_VdpCmR_00007]	Definition of EcucIntegerParamDef VdpCmRTxBufferSize
[ECUC_VdpCmR_00008]	Definition of EcucIntegerParamDef VdpCmRTxAutoTransmission Threshold
[ECUC_VdpCmR_00009]	Definition of EcucFloatParamDef VdpCmRMainFunctionPeriod
[ECUC_VdpCmR_00010]	Definition of EcucIntegerParamDef VdpCmRMaxRelTimestampLen
[ECUC_VdpCmR_00011]	Definition of EcucIntegerParamDef VdpCmRMaxDcaRemDpDCfg Len
[ECUC_VdpCmR_00012]	Definition of EcucBooleanParamDef VdpCmRDevErrorDetect
[ECUC_VdpCmR_00013]	Definition of EcucFloatParamDef VdpCmRMinTxDistanceTime
[ECUC_VdpCmR_00014]	Definition of EcucIntegerParamDef VdpCmRRxBufferSize
[ECUC_VdpCmR_00015]	Definition of EcucReferenceDef VdpCmRStbMSynchronizedTime BaseRef
[ECUC_VdpCmR_00016]	Definition of EcucReferenceDef VdpCmRNvmBlockDescriptor
[ECUC_VdpCmR_00018]	Definition of EcucParamConfContainerDef VdpCmRDcaRemote
[ECUC_VdpCmR_00019]	Definition of EcucIntegerParamDef VdpCmRDcaRemoteId
[ECUC_VdpCmR_00022]	Definition of EcucParamConfContainerDef VdpCmRCslRxPdu
[ECUC_VdpCmR_00023]	Definition of EcucIntegerParamDef VdpCmRCslRxHandleId
[ECUC_VdpCmR_00024]	Definition of EcucReferenceDef VdpCmRCslRxPduRef
[ECUC_VdpCmR_00025]	Definition of EcucParamConfContainerDef VdpCmRCslTxPdu
[ECUC_VdpCmR_00026]	Definition of EcucIntegerParamDef VdpCmRCslTxHandleId
[ECUC_VdpCmR_00027]	Definition of EcucReferenceDef VdpCmRCslTxPduRef
[ECUC_VdpCmR_00028]	Definition of EcucParamConfContainerDef VdpCmRCslSecuredTx Pdu
[ECUC_VdpCmR_00029]	Definition of EcucIntegerParamDef VdpCmRCslSecuredTxHandle Id

▽

△

Number	Heading
[ECUC_VdpCmR_00030]	Definition of EcucReferenceDef VdpCmRCslSecuredTxPduRef

Table C.1: Added Specification Items in R25-11

C.1.2 Changed Specification Items in R25-11

none

C.1.3 Deleted Specification Items in R25-11

Number	Heading
[SWS_XYZ_00001]	SpecItem Title
[SWS_XYZ_00004]	Specification of Production Error <MODULE>_E_<DESCRIPTIVE_NAME>[_INSTANCE>]
[SWS_XYZ_00005]	Specification of Extended Production Error <MODULE>_E_<DESCRIPTIVE_NAME>[_INSTANCE>]

Table C.2: Deleted Specification Items in R25-11

C.1.4 Added Constraints in R25-11

Number	Heading
[SWS_VdpCmR_CONSTR_00001]	Restriction for `VdpCmRCslTxPdu.VdpCmRCslTxPduRef`
[SWS_VdpCmR_CONSTR_00002]	Restriction for `VdpCmRCslRxPdu.VdpCmRCslRxPduRef`

Table C.3: Added Constraints in R25-11

C.1.5 Changed Constraints in R25-11

none

C.1.6 Deleted Constraints in R25-11

none