VECTOR

MICROSAR Classic DCM

Technical Reference

Version 21.05.00

Document Information

History

Author	Date	Version	Remarks
visdth, vissa	2012-08-15	1.00.00	Initial version
vissa	2012-09-21	1.01.00	**Added:** *ReadDataByPeriodicIdentifier (0x2A)* *4.23 InputOutputControlByIdentifier (0x2F)* *5.5.3.8 ReturnControlToECU()* *5.5.3.9 ResetToDefault()* *5.5.3.10 FreezeCurrentState()* *5.5.3.11 ShortTermAdjustment()* *7.8 How to Jump into the FBL From Service DiagnosticSessionControl (0x10)* *8.10 How to Put DCM in a Non-Default Session at ECU Power-On* **Modified:** *Table 5-120 DataServices_<DataName>* *Table 2-7 DET Service IDs*
vissa	2012-12-12	1.02.00	**Added:** *4.15 ReadMemoryByAddress (0x23)* *4.21 DynamicallyDefineDataIdentifier (0x2C)* *4.29 WriteMemoryByAddress (0x3D)* *Table 5-76 Dcm_ReadMemory()* *Table 5-77 Dcm_WriteMemory()* **Modified:** *Table 5-83 ConditionCheckRead(),* *Table 5-84 ReadDataLength(),* *Table 5-88 WriteData() (dynamic length),* *Table 5-89 WriteData() (static length),* *Table 5-90 ReturnControlToECU(),* *Table 5-91 ResetToDefault(),* *Table 5-92 FreezeCurrentState(),* *Table 5-93 ShortTermAdjustment()* -“OpStatus”-parameter availability limitation *Table 5-69 <Module>_<DiagnosticService>()*
vissa	2013-04-17	1.03.00	No changes
vissa	2013-06-28	1.04.00	**Added:** Chapters for OBD service 0x01- 0x0A. *5.6.1.2.6 DtrServices* *5.6.1.2.7 RequestControlServices_<TIDName>*

			5.6.1.2.8 InfotypeServices_<VEHINFODATA> *8.11 How to Support Calibratable Configuration Parameters* **Modified:** *Table 2-2 Not supported AUTOSAR standard conform features* – Removed not supported OBD. *Table 2-6 Limitations to AUTOSAR* – Removed not supported OBD.
vissa	2013-08-20	1.05.00	**Added:** *5.6.1.2.9 CallbackDCMRequestServices_<SWC>* *8.12 How and When to Configure Multiple Protocols* *2.1.1 Deviations* – Added deviation to *CallbackDCMRequestServices_<SWC>* service port. *Table 2-6 Limitations to AUTOSAR* – Added maximum number of supported protocols. – Added maximum number of concurrent client diagnostic connections. **Modified:** *4.14 ReadDataByIdentifier (0x22)* *4.23 InputOutputControlByIdentifier (0x2F)* – Modified configuration and implementation aspects. *Table 5-125 DtrServices_<MIDName>_<TIDName>* *Table 5-126 RequestControlServices_<TIDName>* – Changed port names according to AR DCM SWS. *Table 5-127 InfotypeServices_<VEHINFODATA>* – Editorial change. *Table 2-6 Limitations to AUTOSAR* – Removed not supported multi-protocol. – Removed not supported multiple buffers.
visdth, vissa	2013-09-17	2.00.00	**Modified:** *2.1.1 Deviations*

			– Removed deviations for DID and RID signals. *2.1 Features* – Removed not supported multi-protocol. – Removed not supported multiple buffers. *5.5.3.13 Start(), 5.5.3.14 Stop(), 5.5.3.15 RequestResults()* – Changed function signatures and descriptions *4.23 InputOutputControlByIdentifier (0x2F)* – More details on how optional CEM is supported by DCM. *5.5.3.11 ShortTermAdjustment()* – Removed statement that CEM is included in the controlOptionRecord.
vissa	2014-01-14	2.01.00	**Added:** *8.14 How to Select DEM-DCM Interface Version* *8.15 How to Support OBD and UDS over a Single* Protocol *8.16 How to Use a User Configuration File* *8.18 How to Know When the Security Access Level Changes* **Modified:** *2.4.1 Split Task Functions* – Added configuration aspects. *Table 3-3 Compiler abstraction and memory mapping* – Added calibration parameter memory sections. *4.11 EcuReset (0x11)* – Added clarification for request rejection while waiting for reset execution. *4.13 ReadDTCInformation (0x19)* – Added support of new sub-functions 0x17-0x19. *4.20 ReadDataByPeriodicIdentifier (0x2A)* – Added feature stop periodic reading on changed state. *4.21 DynamicallyDefineDataIdentifier (0x2C)* – Added feature clear DDID on changed state.
vissa, visprk	2014-04-14	2.02.00	**Added:** *9.17 How to Deal with the PduR AR version* **Modified:** *Figure 1-2 Interfaces to adjacent modules of the DCM* – NvM added to figure

			2.4.1 Split Task Functions – Reworked chapter – Added support by configuration tool *4.14.4 Configuration Aspects* – Added information about NvRam signal configuration *4.22.4 Configuration Aspects* – Added information about NvRam signal configuration *5.3 Services used by DCM* – Added NvM services used by the DCM *5.4.3.2 Dcm_StartOfReception(), 5.4.3.4 Dcm_TpRxIndication(), 5.4.3.6 Dcm_TpTxConfirmation()* – New version of the APIs for AR 4.1.2 PduR added *2.1.3 Limitations* – Shared signals between DIDs not supported
vissa	2014-10-08	3.00.00	**Added:** *4.16 ReadScalingDataByIdentifier (0x24)* *5.5.3.12 GetScalingInformation()* *5.6.2 Managed Mode Declaration Groups* *8.19 Post-build Support* **Modified:** *3.1.2 Dynamic Files* – Added post-build data files – Added SWC template file *Table 3-3 Compiler abstraction and memory mapping* – Added post-build data memory mapping *3 Integration* – Added post-build configuration and other BSW headers files *4.11 EcuReset (0x11)* – Sub-functions are now allowed to be user defined. *4.13 ReadDTCInformation (0x19)* – Added new implementation aspect. *5.6.1.2.1 DataServices_<DataName>* – Added new port operation *GetScalingInformation()* *2.1.3 Limitations* – Added limitation for support of DIDranges *5.2.1.1 Dcm_Init()* *6.1 Configuration Variants* – Added new post-build variants

			8.18 How to Know When the Security Access Level Changes – Added link to the corresponding mode declaration group.
vissa, visvkr, visdth	2015-01-13	3.01.00	**Added:** *9.20 DCM AR Version Specific Features* *5.5.3.25 IsDidAvailable()* *5.5.3.26 ReadDidData()* *5.5.3.27 WriteDidData()* *8.20 Handling with DID Ranges* *8.21 How to Support DID 0xF186* **Modified:** *4.14.4, 4.24* – Removed WWH-OBD only DIDs/RIDs from examples. *5.3 Services used by DCM* – AR3 support added *5.4.2 ComM, 5.4.3* PduR – AR3 support added *2.1.3 Limitations* – Removed limitation for not supported DID ranges. – Added limitation for DidRanges. *9.17 How to Deal with the PduR AR version* – Added AR 3.x compliance aspect *Table 2-5 Additions/ Extensions to AUTOSAR* – Added AR 3.x integration
visvkr, vissa	2015-02-06	4.00.00	**Added:** *5.2.1.6 Dcm_InitMemory()* *5.2.3.1 Dcm_GetTesterSourceAddress()* *5.4.4 CanTp* *5.5.1.1<Diagnostic Session Change Notification Callback>* *5.5.1.2<Security Access Change Notification Callback>* *8.22 How to Suppress Responses to Functional Addressed Requests* *8.23 How to Support Interruption on Requests with Foreign N_TA* *8.17 How to Know When the Diagnostic Session Changes* **Modified:** Minor editorial changes

			Table 3-3 Compiler abstraction and memory mapping – Added DCM_VAR_INIT memory section. *9.17 How to Deal with the PduR AR version* – PduR 4.0.1 added *5.2.1.5 Dcm_GetVersionInfo()* – Specified the digit format of the module's version information. *3 Integration* – New include structure – New files introduced *4.17.4 Configuration Aspects* – Added support for single/multiple instace attempt counter, delay timer. *8.18 How to Know When the Security Access Level Changes* – Added new notification type
visade, vissa	2015-05-04	4.01.00	**Added:** *5.2.2.5 Dcm_GetSecurityLevelFixedBytes()* *5.4.3.1 Dcm_TriggerTransmit()* *5.4.3.7 Dcm_TxConfirmation()* *5.5.3.28 GetSecurityAttemptCounter()* *5.5.3.29 SetSecurityAttemptCounter()* *8.24 How to Save RAM using Paged-Buffer for Large DIDs* *8.25 How to Get Security-Access Level Specific Fixed Byte* Values *8.26 How to Extend the Diag Keep Alive Time during Diagnostics* *8.27 How to Recover DCM State Context on ECU Reset/Power On* **Modified:** Global minor editorial changes *Table 4-26 Service 0x19: Supported subservices* – Added sub-function 0x42 *4.10.4 Configuration Aspects* – Added session specific timings aspect *4.11 EcuReset (0x11)* – Added note for resetting to default session if needed. *4.17 SecurityAccess (0x27)* – Additions for the new features listed related to this service *4.18.4 Configuration Aspects;* *4.13.4 Configuration Aspects*

			– Added a hint for external sub-function implementation. *4.23.4 Configuration Aspects* – Removed limitation to static length IO DID for service 0x2F. *4.31.4 Configuration Aspects* – Added missing related configuration parameters *Table 5-119 DCMServices* – Added new operation *Table 2-7 DET Service IDs* – Completed API list – Names converted to hyperlinks for convenience
vissa	2015-11-12	5.00.00	**Added:** *5.2.3.2 Dcm_ProcessVirtualRequest()* *5.2.3.3 Dcm_SetSecurityLevel()* *5.2.2.6 Dcm_SetActiveDiagnostic()* *5.5.2.11 Dcm_FilterDidLookUpResult()* *5.5.2.12 Dcm_FilterRidLookUpResult()* *8.28 How to Define a Diagnostic Connection without USDT* Responses *8.29 How to Handle Multiple Diagnostic Service Variants* **Modified:** Minor editorial changes *4 Diagnostic Service Implementation* – Modified introduction on how to read each diagnostic service sub-chapter. – Added information for the supported types of diagnostic service processor implementations for all services. *4.17 SecurityAccess (0x27)* – Added external service implementation aspect. *8.11.1 OBD Calibration* – Added information about feature configuration. *4.23 InputOutputControlByIdentifier (0x2F)* – More events monitored for automatic IO DID resetting. *Table 2-7 DET Service IDs* – Added missing DET SIDs *5.5.3.25 IsDidAvailable()* – Removed limitation of synchronous usage. *Table 3-3 Compiler abstraction and memory mapping*

			– Added DCM_VAR_INIT memory section for 32bit data.
vissa	2016-03-01	5.01.00	**Added:** *4.32 ResponseOnEvent (0x86)* *5.2.3.4 Dcm_DemTriggerOnDTCStatus()* *5.2.2.7 Dcm_GetRequestKind()* **Modified:** Minor editorial changes. *Table 8-19* – Added details for “RID operation”. *Table 5-119 DCMServices* – Corrected supported return error codes – Added new operations *4.23 InputOutputControlByIdentifier (0x2F)* *5.5.3.8 ReturnControlToECU()* *5.5.3.9 ResetToDefault()* *5.5.3.10 FreezeCurrentState()* *5.5.3.11 ShortTermAdjustment()* – Reworked for support of bitmapped IO DIDs. *Table 2-7 DET Service IDs* – Added new services *2.1.2 Additions/ Extensions* – Added multi byte external CEMR handling *2.1.3 Limitations* – Removed limitation for API *IsDidAvailable()*
vissa	2016-04-29	5.02.00	**Modified:** Minor editorial changes.
vissa	2016-07-05	7.00.00	**Added:** *8.30 How to Switch Between OBD DTR Support by DCM and DEM* *8.31 How to Enable Support of OBD VIDs with Dynamic Length* *9.2 Code Generation Time Messages* **Modified:** Minor editorial changes. *4.8.4 Configuration Aspects* – Reordered FAQ and added variable data size specific hints. *Table 5-48 Services used by the DCM* – Updated used DEM API. *4.14.4 Configuration Aspects* – *Added configuration aspects for OBD MID DIDs.* *4.18.4 Configuration Aspects*

			– Added FAQ for “AllNetworks” parameter. *4.32 ResponseOnEvent (0x86)* – Improved configuration aspects. *5.5.3.23 GetInfotypeValueData()* – Added AR4.2.2 compatibility. *8.19.1.2 Diagnostic Services Part* – Enabled PBS for diagnostic service. *Table 9-1 Compile time error messages* – Added new messages. *Table 2-4 Deviations to AUTOSAR* – Removed deviation to AR for suppression of NRC 0x7E and 0x7F, since AR 4.2.2 now does require this behavior.
vissa	2016-09-22	7.01.00	**Added:** *8.32 How to setup DCM for Sender-Receiver Communication* *8.33 How to Support Routine Info Byte with UDS* **Modified:** Replaced all used DCM_E_OK / DCM_E_NOT_OK to E_OK resp. E_NOT_OK as per [1]. – Note: This is not an API change, since the DCM_E_* symbols have identical values with the standard E_*. You can still use the DCM_E_* ones in your application, if preffered. *8.1 How to Reduce RAM Usage* – Added information regarding DCM buffer size recommendations integrated in the Configuration 5 tool. *5.5.3.23 GetInfotypeValueData()* – Corrected OpStatus parameter description. *10.2 Abbreviations* – Added “C/S” and “S/R”.
visvkr, vissat, visstk	2017-03-20	7.02.00	**Added:** *5.1.3 Dcm_VsgIdentifierType* *5.1.4 Dcm_VsgStateType* *5.2.2.8 Dcm_VsgSetSingle()* *5.2.2.9 Dcm_VsgSetMultiple()* *5.2.2.10 Dcm_VsgIsActive()* *5.2.2.11 Dcm_VsgIsActiveAnyOf()* *8.14.1 Setting the ClientId for DEM AR 4.3.0 and* AR 4.3.1 API *8.22 How to Suppress Responses to Functional Addressed Requests* *8.23 How to Support Interruption on Requests with Foreign N_TA*

			8.25.3 Security Level Fixed Bytes variant handling with VSGs *8.34 Vehicle System Group Support* **Modified:** *5 API Description* – Corrected Particularities and Limitations. *5.1.2 Dcm_RecoveryInfoType* – Added active protocol member. *5.3 Services used by DCM* – Added Dem_[Dcm]ClearDTC. – Added new DEM AR 4.3.0 APIs. *5.6.1.1.1 DCMServices* – Added VSG services. *8.11.1.2 Calibration of Supported OBD Parameter Identifier* – Corrected calibratable configuration symbols. *8.14 How to Select DEM-DCM Interface Version* – Also mention DEM AR 4.3.0 APIs.
vissa, visvkr	2017-05-16	8.00.00	**Added:** *Table 5-69 <Module>_<DiagnosticService>()* **Modified:** *5.5.2.1 <Module>_<DiagnosticService>()* *5.5.2.2 <Module>_<DiagnosticService>_<SubService>()* – Restructured chapters to describe API variation depending on licensed DCM AR version. – Added information for pMsgContext and job finishing activities. *Table 2-8 Errors reported to DET* – Added DCM_E_CRITICAL_ERROR.
viswsi, visaey, visvkr	2017-06-06	8.02.00	**Added:** *5.6.3 Sender-Receiver Interface* **Modified:** *8.32.1 Implementation Limitations* – Added new data types for Sender-Receiver communication. *3.3 Compiler Abstraction and Memory Mapping* – Updated to ASR 4.0.3 naming convention.
visstk, viswsi, visaey	2017-06-13	8.03.00	**Modified:** *5.3 Services used by DCM*

			– BswM_Dcm_ApplicationUpdated is not only available in AUTOSAR 4.x but also AUTOSAR 3.x. *8.32.1 Implementation Limitations* – Added Boolean data type. 5.6.3 *Sender-Receiver Interface* *4.24.3 Implementation Aspects* – Added information for routine management on session transition.
visstk, visaey	2017-06-30	8.04.00	**Added:** *5.1.5 Dcm_InputOutputControlParameterType* *5.1.6 Dcm_IOOperationResponseType* *5.1.7 Dcm_IOOperationRequest_< DidName/DidDataName >Type* **Modified:** *4.20.1 Functionality* - Added UUDT DelayTimer *4.20.3 Implementation Aspects* - Added UUDT DelayTimer *4.20.4 Configuration Aspects* - Added UUDT DelayTimer
viswsi	2017-08-02	8.05.00	**Modified:** Minor improvements. *4.17.3 Implementation Aspects* - Updated implementation aspects regarding external sub-services.
viswsi	2017-09-18	8.06.00	**Added:** *8.35 Usage Hints for Operation with SilentBSW*
viswsi	2017-09-29	9.00.00	**Modified:** *5.6.3 Sender-Receiver Interface* *8.32.3 Configuration Aspects* - Added PR-Port type
visuvo	2017-11-22	9.01.00	**Modified:** *4.20.1 Functionality* *4.20.3 Implementation Aspects* *4.20.4 Configuration Aspects* - Added adaption of maximum UUDT Frame size
visvkr	2017-12-18	9.02.00	**Modified:** *8.1 How to Reduce RAM Usage* - Paged-Buffer for DID *8.24 How to Save RAM using Paged-Buffer for Large DIDs* - Dedicated Paged-Buffer configuration
visstk,	2018-01-15	9.03.00	**Modified:**

visahe			*3.1.1 Static Files* - Added Base files description *3 Integration* - Added include structure modifications (Base files added) *4.17.3 Implementation Aspects* - Document deviation to [10] regarding power-on delay timer *9.20 DCM AR Version Specific Features* - Added missing sub-chapter *5.2.3.1 Dcm_GetTesterSourceAddress()* - Defined return code more precisely
visahe, viswsi	2018-02-02	9.04.00	**Modified:** *4.32 ResponseOnEvent (0x86)* - Added onChangeOfDataIdentifier description *8.32.2 Application usage Scenario* - Added a caution hint regarding “ReturnControlToECU”
visahe, visygr	2018-02-27	9.05.00	**Modified:** *8.32 How to setup DCM for Sender-Receiver Communication* - Added reference to the Application Note for the diagnostic transformer usage
viswsi, visuvo	2018-04-18	10.00.00	**Added:** *8.35.3 Automatically Enabled Features* **Modified:** *4.30.4 Configuration Aspects* - Updated caution hint regarding external service implementation of service 0x3E
visuvo, visstk	2018-05-14	10.01.00	**Removed:** *8.35 Subchapter: Automatically Disabled Features* **Modified:** *8.1.3 Exchanging data between OS tasks* - Intermediate buffer replaced by improved size checks - Updated name of configuration parameter for implicit communication *8.24 How to Save RAM using Paged-Buffer for Large DIDs* - Update effect of page size configuration
viswsi	2018-06-19	10.02.00	**Added:** *5.5.2.13 Dcm_HandleServiceExtern()*

			8.36 How to Support Diagnostic Service Dispatching
visuvo	2018-07-20	10.03.00	**Modified:** *8.24.3 Implementation Limitations* - Implementation limitations for VIDs (0xF800-0xF8FF) added
visahe	2018-08-29	10.04.00	**Modified:** *8.1.3 Exchanging data between OS tasks* - Info box added. *8.35 Usage Hints for Operation with SilentBSW* - Minor improvements.
visstk	2018-10-01	11.00.00	**Modified:** *4.17.4 Configuration Aspects* - Attempt counter reset parameter added
visvkr	2018-10-23	11.01.00	**Modified:** - Deprecated interfaces removed
visygr	2018-12-11	11.03.00	**Added:** *5.1.8 Dcm_SpecificCauseCodeType* *5.2.2.12 Dcm_SetSpecificCauseCode()* *8.37 How to provide an additional byte (specific cause code)* to negative responses
visahe	2019-01-25	11.05.00	**Modified:** *8.24.3 Implementation Limitations*
visahe	2019-03-12	11.06.00	**Added:** *8.39 How to Support Parallel Service Processing*
viswse	2019-05-02	12.01.00	**Added:** *8.38 How to Deactivate S3 Timer*
visygr, visahe	2019-05-21	12.02.00	**Added:** *8.39 How to Configure Generic Connections* **Modified:** 5.1.2 *Dcm_RecoveryInfoType* 5.2.2.7 *Dcm_GetRequestKind()* 5.2.3.1 *Dcm_GetTesterSourceAddress()* 5.2.3.2 *Dcm_ProcessVirtualRequest()* 8.10 *How to Put DCM in a Non-Default Session at ECU Power-On* **Removed:** Misleading sections and chapters
visahe, viswse	2019-08-02	12.05.00	**Added:** 8.40 How to Persist Dynamic Defined DIDs
visvkr	2019-09-12	13.00.00	**Added:** *4.13 ReadDTCInformation (0x19)* - Support sub-function 0x55 **Modified:** - Major format improvements

			- Tables in *8.12.1 Diagnostic Client(s) Processing Prioritization* improved
vsarcbada, vsarcafro, vsarcmeba, visygr	2019-10-01	13.01.00	**Added:** *4.19 Authentication (0x29)* *5.4.5 KeyM* *5.4.6 Csm* *5.6.2.7 DcmAuthenticationState* *8.41 How to Authenticate* **Modified:** *1.2 Architecture Overview* - Updated *Figure 1-2 Interfaces to adjacent modules of the DCM* with KeyM and Csm *2.3 States* - Added the authentication state *5.3 Services used by DCM* - Added the used services of Csm and KeyM
visvkr	2019-11-08	13.02.00	**Modified:** - Minor corrections for *Authentication (0x29)*
visepp, visvkr	2019-12-04	13.03.00	**Modified:** *5.4.6.1 Dcm_CsmAsyncJobFinished()* - Signature changed *8.15 How to Support OBD and UDS over a* Single Protocol - Feature extended for single protocol *8.12.1 Diagnostic Client(s) Processing Prioritization* - Tables improved
visepp	2020-01-30	13.04.00	**Added:** *5.2.2.13 Dcm_SetSecurityBypass()* *8.42 How to Bypass Pre-Condition Checks* **Modified:** *8.40 How to Persist Dynamic Defined DIDs* - Added configuration aspects
visepp	2020-02-28	13.05.00	**Added:** *8.43 Supported Data Types for DIDs and RIDs*
visvkr	2020-04-14	14.00.00	**Added:** - Extended features for *Authentication (0x29)*
visepp	2020-04-24	14.01.00	**Added:** *1.3 Legal Information*
visepp	2020-05-19	14.02.00	**Modified:** *5.1.6 Dcm_IOOperationResponseType* - Removed DCM_IDLE value
visepp	2020-06-10	14.03.00	**Modified:** *8.14 How to Select DEM-DCM Interface Version* - Added DEM AR 4.3.1 API support
visuvo	2020-06-25	14.04.00	**Added:**

			4.13 ReadDTCInformation (0x19) - Support sub-function 0x16
visepp, visuvo, visvkr	2020-09-02	15.00.00	**Added:** *4.13 ReadDTCInformation (0x19)* - Support sub-function 0x56 *4.25 RequestDownload (0x34)* *4.27 TransferData (0x36)* *4.28 RequestTransferExit (0x37)* *Table 5-114 Dcm_ProcessRequestDownload()* *Table 5-116 Dcm_ProcessTransferDataWrite()* *Table 5-118 Dcm_ProcessRequestTransferExit()* **Modified:** *5.1.2 Dcm_RecoveryInfoType* - Removed Exist-Condition for struct element Active Protocol
visepp	2020-10-28	15.02.00	**Modified:** *5.5.3.34 ProcessRequestTransferExit()* - Added arguments
viswse, visvkr, visahe	2020-12-10	15.05.00	**Added:** *4.24.5 Request and Response Data Pointers in an Operation for RIDs* **Modified:** *3.1.1 Static Files* *4.32 ResponseOnEvent (0x86)* - Removed irrelevant information
visygr	2021-01-12	15.06.00	No relevant changes
visygr, viswse, vsarcmiem, visvkr	2021-02-24	15.08.00	**Added:** *4.13 ReadDTCInformation (0x19)* - Support sub-function 0x1A **Modified:** *Table 2-2 Not supported AUTOSAR standard conform features* - Add unsupported AUTOSAR features *5.2.2.2 Dcm_GetSecurityLevel()* - Note added in functional description *5.3 Services used by DCM* - Added the used service of sub-function 0x1A *8.26 How to Extend the Diag Keep Alive Time during Diagnostics* - Configurable behavior for TesterPresent requests *4.32 ResponseOnEvent (0x86)* - Update to ISO 14229-1 (2020)
visepp	2021-03-26	16.00.00	**Modified:** Corrected some API descriptions in: *5.2 Services provided by DCM*

			5.5 Configurable Interfaces
visepp	2021-05-17	16.02.00	**Modified:** *Table 2-2 Not supported AUTOSAR standard conform features* - Removed restriction for array data types for routines.
visvkr, viswse, visygr	2021-06-08	16.03.00	**Added:** *4.14 ReadDataByIdentifier (0x22)* - Limitation added for OBDonUDS *5.3 Services used by DCM* - Renaming GetNumberOfFilteredExtendedDataRecords to GetSizeOfFilteredExtendedDataRecords *8.44 How to Configure DIDs with static content*
visahe, visvkr	2021-06-30	16.04.00	**Added:** *8.45 How to report security events to IdsM* **Modified:** *4.32 ResponseOnEvent (0x86)* - Updates related to ISO 14220-1 (2020)
visvkr	2021-08-02	16.05.00	**Modified:** *8.45 How to report security events to IdsM* - Simplification of the configuration *4.32 ResponseOnEvent (0x86)* - Updates related to ISO 14220-1 (2020)
dng	2021-09-13	17.00.00	**Modified:** *4.11 EcuReset (0x11)* - Support of a positive response after reset
dng, visuvo	2021-10-08	17.01.00	**Added:** *How to Support Parallel UDS/UDS Services in Default Session* **Modified:** *4.13 ReadDTCInformation (0x19)* - Updates related to Fault Memory Records *4.19 Authentication (0x29)* - Implementation Aspects *4.32 ResponseOnEvent (0x86)* - Updates related to ISO 14220-1 (2020)
viswse	2021-12-08	17.03.00	**Modified:** *4.24.5 Request and Response Data Pointers in an Operation for RIDs* - Parallel Processing of Service 0x31 in Default Session *How to Support Parallel UDS/UDS Services in Default Session*

			- Services 0x19 and 0x31 added
visahe visuvo visvkr visepp	2022-01-25	17.04.00	**Modified:** *4.12.4 Configuration* Aspects - Support user defined fault memories for service 0x14 added *4.32 ResponseOnEvent (0x86)* - Added onComparisonOfValues (sub-function 0x07) description - Added storageState “storeEvent” *4.19 Authentication (0x29)* - Added sub-function 0x04 (transmitCertificate) *3 Integration* - Added Dcm_MemMap.arxml and Dcm_MemMap.h
visvkr visahe alefarth	2022-02-18	17.05.00	**Added:** *8.13 How to Select DCM AUTOSAR Version* **Modified:** Product name updated to MICROSAR Classic *4.12.4 Configuration* Aspects - Support authentication check for user defined fault memories for service 0x14 *4.32 ResponseOnEvent (0x86)* - Adaption according to ISO 14229-1 (2020)
visepp visvkr	2022-03-31	18.00.00	**Modified:** *How to Support Parallel UDS/UDS Services in Default Session* - Service 0x3E added *8.32 How to setup DCM for Sender-Receiver Communication* - Supported data types extended
viswse dng	2022-04-13	18.01.00	**Added:** *8.46 How to Support Secure Coding Features* **Modified:** *8.39 How to Support Parallel Service Processing* - Handling of parallel processing for OBD/UDS and UDS/UDS described in one chapter - Some more services added for UDS/UDS **Removed:** *How to Support Parallel UDS/UDS Services in Default Session*
dng, visepp	2022-05-18	18.02.00	**Modified:** *5.5.3 Required Port Operation Functions* - Data types of functions for reading and writing data changed *8.43 Supported Data Types for DIDs and RIDs*

			- Support to Sender-Receiver Interface and DCM Function Callout Interface added *3.1.1 Static Files* - Removed Dcm_<Unit>Int.h *3 Integration* - Updated include structure
visepp, visvkr	2022-06-23	18.04.00	**Modified:** *4.19.4 Configuration Aspects* - Added more detailed information about the configuration of the Crypto Service Manager *8.39 How to Support Parallel Service Processing* - Updated and moved to *8.12.2 Diagnostic Client(s) Processing Parallelization* *1 Introduction* - J1979-2 OBDonUDS mentioned
visuvo, visepp, dng	2022-08-12	18.05.00	**Modified:** *4.11.1 Functionality* - Removed sentence about communication reaction *4.11.4 Configuration Aspects* - Removed configuration parameter DcmDslDiagRespOnSecondDeclinedRequest *4.19 Authentication (0x29)* - Support of User Specific Authentication Roles *5.5.2 Callout Functions* - Added the required function for supporting User Specific Authentication Roles *8.19.1.2.1 Handling of State Execution Preconditions of Variant Diagnostic Entities* - Reworked examples and description
visvkr	2022-09-16	19.00.00	**Modified:** Minor modifications and corrections.
dng	2022-11-08	19.02.00	**Modified:** *5.5 Configurable Interfaces* - Adaptation according to AUTOSAR version R19-11 *8.10 How to Put DCM in a Non-Default Session at ECU Power-On* - Changed the preconditions of providing ConnectionId *8.13 How to Select DCM AUTOSAR Version* - Added new changes to DCM in AUTOSAR version R19-11
visepp	2022-12-05	19.03.00	**Modified:** *4 Diagnostic Service Implementation*:

			- Added information about ability to temporarily disable services
visepp, visvkr, dbusch, dng	2023-02-13	19.05.00	**Added:** *4.32 ResponseOnEvent (0x86)* - Added ReportDTCRecordInformationOnDtcStatusChange (sub-function 0x09) description **Modified:** *8.13 How to Select DCM AUTOSAR Version* - Clarifications *5.2.2.1 Dcm_GetActiveProtocol()* - Clarifications *8.8 How to Handle Characteristics at Session Change with Switch to/from FBL and ECU Reset* - Reworked section and added documentation on ChannelReady feature *8.43 Supported Data Types for DIDs and RIDs* - Reworked section
dbusch, dng	2023-03-14	20.00.00	**Added:** *4.26 RequestUpload (0x35)* **Modified:** *4.27 TransferData (0x36)* *3.1.1 Static Files* - Removed files for the Dcm extension part
dbusch, dng	2023-05-24	20.02.00	**Added:** *8.47 How to Use Diagnostic Service Pre-*/Post-Handler **Modified:** *3.1.1 Static Files* - Removed files for the Dcm core part
dbusch, visvkr	2023-06-05	20.03.00	**Modified:** Minor modifications and corrections. *4.19 Authentication (0x29)* - Clarifications regarding Non-/Generic Connections
dbusch, visuvo	2023-08-16	20.05.00	**Added:** *8.48 How to Use Protocol specific restriction for DIDs and RIDs* **Modified:** *8.1 How to Reduce RAM Usage* - Clarifications regarding minimum buffer size *8.44 How to Configure DIDs with static content* - Use of padding bytes added *5.5.2.1 <Module>_<DiagnosticService>()* - Use of Dcm_ExtendedOpStatusType for AUTOSAR Version equal or newer than R19-11

			5.5.2.2 <Module>_<DiagnosticService>_<SubService>() - Use of Dcm_ExtendedOpStatusType for AUTOSAR Version equal or newer than R19-11
visvkr, dbusch	2023-09-29	21.01.00	**Added:** *7 Cybersecurity* *5.1.9 Dcm_AuthenticationInfoType and following* *5.2.3.5 Dcm_DeauthenticateConnection()* *5.2.3.6 Dcm_AuthenticateConnection()* *5.5.1.3 Dcm_ConnectionAuthenticated()* **Modified:** *4.19 Authentication (0x29)* - Clarifications regarding external APIs to set authentication states *8.41 How to Authenticate* - Clarifications regarding external APIs to set authentication states
mosman, visepp	2023-11-15	21.03.00	**Added:** *7.2.6 CMI-Dcm-AuthenticationBypass* *8.42 How to Bypass Pre-Condition Checks* **Modified:** *4.17 SecurityAccess (0x27)* - Variance for key, seed, and ADR size
visvkr, mosman	2023-12-20	21.04.00	**Modified:** *7 Cybersecurity* - TCR-IDs replaced *4.15 ReadMemoryByAddress (0x23)* and *4.29 WriteMemoryByAddress (0x3D)* - Numerical and symbolic representation of memory ranges
visepp dbusch	2024-02-14	21.05.00	**Added:** *8.49 How to prevent a security reset at a session self-transition* *4.32.3.5 Pause RoE Processing* *5.2.3.7 Dcm_SetRoeActivationState()* **Modified:** *4.19 Authentication (0x29)* *5.2.3.5 Dcm_DeauthenticateConnection()* *8.41 How to Authenticate* - Update regarding API availability

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Diagnostic Communication Manager	R4.2.2
[2]	AUTOSAR	Specification of Diagnostic Communication Manager	R4.3.0
[3]	AUTOSAR	Specification of Diagnostic Communication Manager	R19-11
[4]	AUTOSAR	Specification of Diagnostic Communication Manager	R20-11
[5]	AUTOSAR	Specification of Diagnostic Communication Manager	R21-11
[6]	AUTOSAR	Specification of Development Error Tracer	R3.2.0
[7]	AUTOSAR	Specification of Diagnostic Event Manager	R4.3.1
[8]	AUTOSAR	List of Basic Software Modules	R1.0.0
[9]	AUTOSAR	Specification of Crypto Service Manager	R19-11
[10]	ISO	ISO 14229-1 UDS	2013
[11]	ISO	ISO 14229-1 UDS	2020
[12]	ISO	ISO 15031-5 OBD	2004
[13]	ISO	ISO 27145-2 WWH-OBD CDD Emissions	2009
[14]	ISO	ISO 27145-3 WWH-OBD CMD	2009
[15]	SAE	SAE J1979-2 OBDonUDS	2021
[16]	Vector	Technical Reference NvM	-
[17]	Vector	User Manual Post Build Selectable	-
[18]	Vector	User Manual Identity Manager	-
[19]	Vector	Technical Reference Diagnostic Event Manager	-
[20]	Vector	Technical Reference CSM	-
[21]	Vector	Technical Reference KeyM	-
[22]	Vector	AN-ISC-8-1218_Atomic_Dcm_S-R_Interfaces_with_Diagnostic_Transformer	-

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

Contents

Illustrations

Tables

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module DCM as specified in [1].

Supported Configuration Variants:	pre-compile, post-build loadable, post-build selectable	
Vendor ID:	DCM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	DCM_MODULE_ID	53 decimal (according to ref. [8])

The AUTOSAR DCM is a software component that

- handles the diagnostic communication between the tester and the ECUs application
- analyzes and interprets multiple diagnostic communication protocols:
  - UDS: ISO 14229 ([10] and [11])
  - OBD: ISO 15031 ([12])
  - WWH-OBD: ISO 27145 ([13] and [14])
  - OBDonUDS: SAE J1979-2 ([15] based on [5]).
- implements the handling of diagnostic services, providing abstract interface to the application by hiding all protocol specifics
- provides a built-in handling of the fault memory manager (DEM) data acquisition
- provides service execution precondition validation and state management such as diagnostic sessions and security access verification as well as custom ECU mode condition verification (e.g. vehicle speed, etc.).

1.1 How to Read This Document

Here are defined some general rules on how to read this document.

1.1.1 DCM Integration and Basic Operation

We recommend starting with the chapter *3 Integration*. It will help you to bind the DCM component into your project and to learn about its integration specific requirements. Once the code binding is finished in your project, please go on with the *Functional Description* chapter to learn about how to operate the DCM.

1.1.2 Diagnostic Service Documentation

Once the DCM is integrated into your project, you will need to know how each diagnostic service, your ECU must support, is to be configured, implemented and handled by DCM and your application. For learning that, please refer to chapter *4 Diagnostic Service Implementation*.

1.1.3 API Definitions

You can any time directly refer to a DCM provided/required service or callout description once you have started the DCM application implementation, by searching for the function name in this document. But the usual way is to start with the usage context of the concrete function you are looking for:

- the diagnostic service it is bound to (investigate the corresponding *Diagnostic Service Implementation* sub-chapter*)*
- a special feature it serves (investigate the corresponding *Using the DCM* “How to…” sub-chapter)

1.1.4 DCM Configuration Parameter Descriptions

This document contains many references to DCM configuration parameters. The goal of this document is not to describe the parameters in detail, but to show you which parameters are bound to which diagnostic services or features. All those parameter references are given as full path links within the DCM DaVinci Configurator 5 for faster location of the concrete parameter. Once you have followed such a link in the DaVinci Configurator 5 tool, please read the description information bound to the parameter. Follow any dependency links from this description to learn more about what additionally shall be configured to get a fully functioning configuration.

1.2 Architecture Overview

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

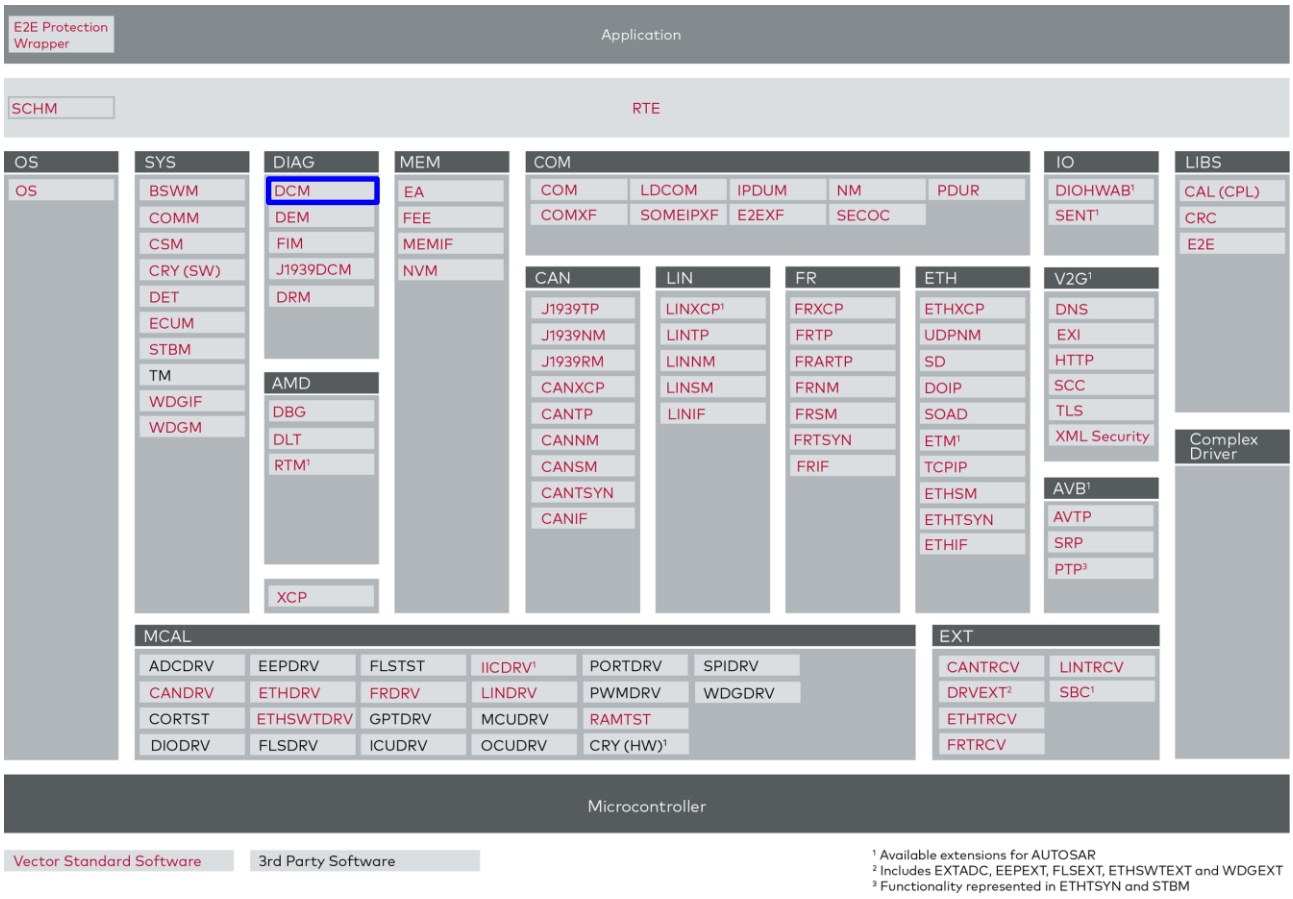

Figure 1-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the DCM. These interfaces are described in chapter 4.

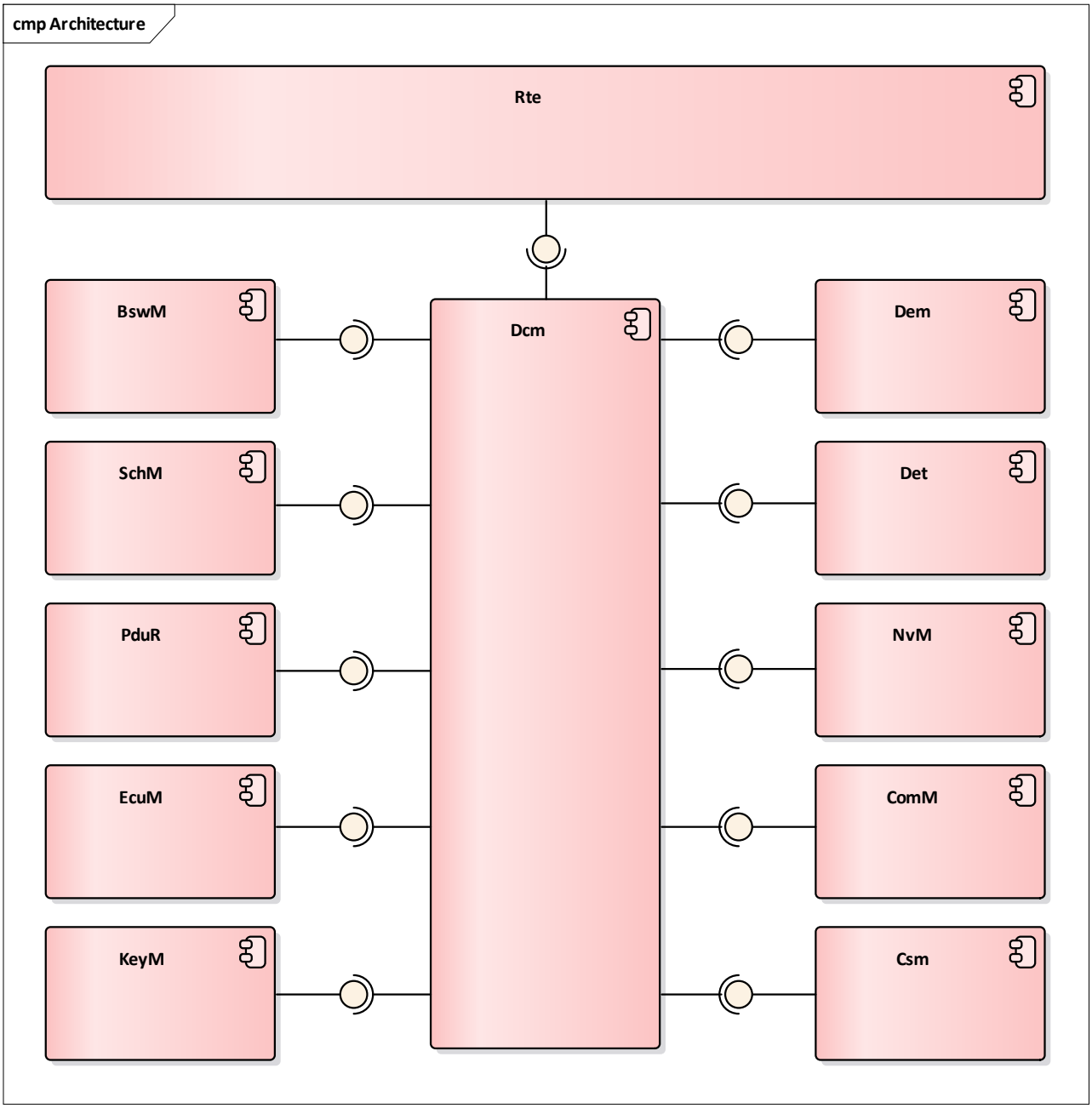

Figure 1-2 Interfaces to adjacent modules of the DCM

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by the DCM are listed in chapter 5.5.3.1 based on their definition in [1]. In some cases where the DCM

requires a call out extension, the DCM calls a CDD module directly through the Dcm_Cdd interface.

1.3 Legal Information

Caution
The DCM is highly configurable and provides a variety of interfaces. It is therefore possible that certain configurations and usage scenarios that the customer plans, intends or specifies do not comply with applicable laws, statutes, regulations and/or standards, in particular, but not limited to, vehicle emission standards (hereinafter collectively "***Legal Requirements***"). It is the sole responsibility of the customer (i) to configure and use the DCM and its interfaces in such a way that implementation and use of the DCM comply with all applicable Legal Requirements, as amended from time to time, and (ii) to take all measures required by such Legal Requirements for the operation and distribution of the customer system in which the DCM is implemented, in particular, but not limited to, obtaining approvals under regulatory procedures prescribed by Legal Requirements.

2 Functional Description

2.1 Features

The features listed in the following tables cover the complete functionality specified for the DCM.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> *Table 2-1 Supported AUTOSAR standard conform features*

> *Table 2-2 Not supported AUTOSAR standard conform features*

Vector provides further DCM functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> *Table 2-3 Features provided beyond the AUTOSAR standard*

The following features specified in [3] are supported:

Supported AUTOSAR Standard Conform Features
MICROSAR Classic Identity Manager using Post-Build Selectable

Table 2-1 Supported AUTOSAR standard conform features

The following features specified in [3] are not supported:

Not Supported AUTOSAR Standard Conform Features
No link time configuration support.
No post-build loadable support on diagnostic services (only communication).
Handling of different diagnostic protocols: API Xxx_StopProtocol() is not supported.
Environmental conditions: Mode conditions with S/R data elements and certificate compare elements are not supported.
Following literals of DcmDspDataUsePort are not supported: > USE_DATA_ASYNCH_FNC_ERROR > USE_DATA_ASYNCH_CLIENT_SERVER_ERROR > USE_DATA_SENDER_RECEIVER_AS_SERVICE > USE_ECU_SIGNAL
Xxx_ReadData callout with ErrorCode parameter is not supported.
"ECU signals" (IoHwAb) are not supported.
BndM DIDs are not supported.
Services 0x38 (RequestFileTransfer) and 0x87 (LinkControl) are not internally supported. For details about supported services, see *4 Diagnostic Service Implementation.*
Service 0x14: Xxx_ClearDTCCheckFnc to check if application allows to clear the DTC is not supported.

Not Supported AUTOSAR Standard Conform Features
Service 0x22: > Composite DIDs using DcmDspDidRef are not supported. > Configuration of support information is not supported.
Service 0x27: SecurityMaxAttemptCounterReadoutTime is not supported.
Service 0x29: Verification of target identification is not supported.
Service 0x2A: > Periodic transmissions using a separate protocol and buffer is not supported. > Periodic transmission type 2 is partially supported. > Explicit resource limitation of connections used for slow, medium and fast transmission is not supported.
Service 0x31: > Access permission evaluation (Session/Security/Mode/Authentication) is only on RID level and not on operation level (Start/Stop/RequestResults) supported. > Xxx_StartConfirmation, Xxx_StopConfirmation and Xxx_RequestResultsConfirmation operations are not supported.
Service 0x23/0x3D: Session restriction evaluation is only on service level and not on memory range level supported.
Service 0x86: > EventWindowTime 0x04 (CurrentAndFollowingCycle) not supported. > Pre-configuration of RoE events is not supported. Transmission of ServiceToRespondTo on a different TxPduId (TYPE 2) than the RoE response is not supported.
OBD Services: > Multi-Signal OBDDataIdentifieres are not supported. > Typed PID access besides of uint8 pointer is not supported. > Configuration of support information is not supported. > Disabling of OBD mirroring is not supported.
Dcm_GetVin() is not supported
For details about not supported configuration parameters see the delivered description file (BSWMD) of DCM.

Table 2-2 Not supported AUTOSAR standard conform features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond the AUTOSAR Standard
Possibility to avoid high CPU load peaks: *How to Reduce DCM Main-Function Run Time Usage*
Optimized multi-client communication support: *How to Handle Multiple Diagnostic Clients Simultaneously*
Runtime optimized DCM-DEM interface for low CPU load.
Native AR 4.1.2 and AR 4.3.0 DEM API version support.
Support for sub-functions 0x17, 0x18 and 0x19 of service *ReadDTCInformation (0x19)* according to [10].
Optional notification on security access level change
Extensible keep-alive time: *How to Extend the Diag Keep Alive Time during Diagnostics* Recovery of DCM states over reset/power down: *How to Recover DCM State Context on ECU Reset/Power*
Restricting the access of DIDs and RIDs to specific protocols: *How to Use Protocol specific restriction for DIDs and RIDs*

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.1 Deviations

Deviation	Statement
CallbackDCMRequestServices_<SWC>	Operation StopProtocol not supported since not fully specified in AR 4.0.3 SWS DCM what a protocol stop really does. Instead a single protocol switch point is realized by *StartProtocol()*.
Usage of NvM_SetBlockLockStatus()	NvM_SetBlockLockStatus() must initially be called with TRUE to ensure, that the SWC is no longer able to update the NvM Block. In the SWS of NvM it is stated, that if NvM_SetBlockLockStatus() was called with TRUE, the NvM Block shall not be writeable till ECU shutdown. This includes write accesses by DCM. But the MICROSAR NvM is able to recognize a write access by DCM due to the special BlockId (NvM_GetDcmBlockId()). Therefore, DCM is still able to update the NvM Block although the NvM_SetBlockLockStatus() was called with TRUE.

Table 2-4 Deviations to AUTOSAR

2.1.2 Additions/ Extensions

Additions/Extensions	Statement
DCM CPU peak load reduction support.	See *8.1.4*
RAM and run time optimization parameters for multi-client support	See *8.4*
Optimized DCM DEM iterator	DCM internal design.
Calibratable configuration parameters	See *8.11*
Used definition for no active protocol (DCM SWS AR 4.1.1): DCM_NO_ACTIVE_PROTOCOL	Required since before the very first diagnostic request is received, there is no active protocol assigned in DCM. But at the same time the *Dcm_GetActiveProtocol()* shall return a valid value.
Support for DEM AR 4.3.0 API	See *8.14*
Combined OBD and UDS protocols over a single client connection	See *8.15*
Notification on security access level state change	See *8.18*
Suppression on functional addressed requests	See *8.22*
Support of paged-buffer data access on DID signals	*See 8.24*
Configurable Security-Access level specific fixed bytes	*See 8.25*
Extensible keep-alive time	*See 8.26*
DCM state recovery on reset/power on	*See 8.27*
Alternative solution for diagnostic service variant handling	*See 8.29*
Support for externally handled CEMR with more than four-byte control mask.	According to [1], a CEMR can be only up to four bytes in size. MICROSAR Classic DCM extends the SWS by also allowing any size of CEMR. Refer to *InputOutputControlByIdentifier (0x2F)* for details.

Table 2-5 Additions/ Extensions to AUTOSAR

2.1.3 Limitations

Limitation	Statement
OEM specific RoE support.	Due to insufficient specification in the DCM SWS, the RoE support can only be implemented for specific OEM requirements.
Support of up to 32 protocols	Required due to optimized implementation of service to protocol mapping.
Support of up to 32 concurrent client connections	Required due to optimized implementation of concurrent request processing.
Sharing of signals between DIDs not supported	Required due to the inability to differentiate between the callers of a signal (e.g. service 0x22 and 0x2A).

Table 2-6 Limitations to AUTOSAR

2.2 Initialization

At ECU power-on boot (or any reset situation) DCM must be initialized by calling the API *Dcm_Init()*. If multiple configuration variants are supported, a concrete configuration variant parameter is expected by this API.

2.3 States

DCM currently manages the following state machines:

- Diagnostic session states (managed by service *DiagnosticSessionControl (0x10)*)
- Security access states (managed by service *SecurityAccess (0x27)*)
- ECU Communication activity (managed by the ComM)
- DTC setting allowance (managed by the Dem)
- Authentication states (managed by service *Authentication (0x29)*)

2.4 Main Functions

In order to function properly, the *Dcm_MainFunction()* must be called periodically in the configured time period.

To specify the DCM task cycle time, set up the configuration parameter:

/Dcm/DcmConfigSet/DcmGeneral/DcmTaskTime

2.4.1 Split Task Functions

2.4.1.1 Functionality

Dcm_MainFunction() is only a container function that calls the two functions *Dcm_MainFunctionTimer()* and *Dcm_MainFunctionWorker()*. Of these two, only the *Dcm_MainFunctionTimer()* depends on a stable cycle time. If you find it difficult to run the *Dcm_MainFunction()* on a high priority task to ensure the timing behavior, you can optionally call these two functions instead of *Dcm_MainFunction()*.

This allows you to run the *Dcm_MainFunctionTimer()* on a higher priority task to guarantee the UDS timing requirements e.g. sending of NRC 'RequestCorrectlyRecieved-ResponsePending'.

Please note, both the *Dcm_MainFunctionWorker()* and *Dcm_MainFunctionTimer()* are optimized for short run time, so this option is usually not necessary.

2.4.1.2 Configuration Aspects

Per default DCM has only one *Dcm_MainFunction()* i.e. has no split tasks as specified in [1]. To enable split task usage in DCM, you must set up DCM in the configuration tool as follows:

> Activate main-function task splitting via parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmSplitTasksEnabled

> Both *Dcm_MainFunctionTimer()* and *Dcm_MainFunctionWorker()* will be scheduled for the time period specified by: /Dcm/DcmConfigSet/DcmGeneral/DcmTaskTime

> Optionally you can specify different scheduling time for the *Dcm_MainFunctionWorker()* using parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmMainFunctionWorkerTaskTime

2.4.1.3 Integration Aspects

Both main-functions are automatically registered for scheduling in SchM component via SWC-template, but still they have no assigned task priority relation. As the *Dcm_MainFunctionTimer()* handles the real-time aspect of the DCM component, it must be running under high OS task priority. The *Dcm_MainFunctionWorker()* shall be assigned to an OS task that has a lower or equal priority compared to the *Dcm_MainFunctionTimer()*'s task.

Caution

> Do **not** assign the *Dcm_MainFunctionWorker()* on a higher priority task than the *Dcm_MainFunctionTimer()*, especially not if your OS supports task preemption.
> You need **both** *Dcm_MainFunctionWorker()* and *Dcm_MainFunctionTimer()* (unless you use the *Dcm_MainFunction()*).

2.5 Error Handling

2.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [6], if development error reporting is enabled (i.e. pre-compile parameter `DCM_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator but must have the same signature as the service `Det_ReportError()`. The reported DCM ID is 53.

The reported service IDs identify the services which are described in *5 API Description*. The following table presents the service IDs and the related services:

Service ID	Service
0x00	*Dcm_StartOfReception()*
0x01	*Dcm_Init()*
0x02	*Dcm_CopyRxData()*
0x03	*Dcm_TpRxIndication()*
0x04	*Dcm_CopyTxData()*
0x05	*Dcm_TpTxConfirmation()*
0x06	*Dcm_GetSesCtrlType()*
0x0D	*Dcm_GetSecurityLevel()*
0x0F	*Dcm_GetActiveProtocol()*
0x21	*Dcm_ComM_NoComModeEntered()*
0x22	*Dcm_ComM_SilentComModeEntered()*
0x23	*Dcm_ComM_FullComModeEntered()*
0x24	*Dcm_GetVersionInfo()*
0x25	*Dcm_MainFunction()*
0x2A	*Dcm_ResetToDefaultSession()*
0x2B	*Dcm_DemTriggerOnDTCStatus()*
0x30	*Dcm_ExternalSetNegResponse()*
0x31	*Dcm_ExternalProcessingDone()*
0x32	*<Module>_<DiagnosticService>()*
0x34	*ReadData() (synchronous)*
0x3B	*ReadData() (asynchronous)*
0x3F	*IsDidAvailable()*
0x40	*ReadDidData()*
0x41	*WriteDidData()*
0x44	*GetSeed() (with SADR)*
0x45	*GetSeed() (without SADR)*
0x47	*CompareKey()*
0x56	*Dcm_SetActiveDiagnostic()*
0x59	*GetSecurityAttemptCounter()*
0x5A	*SetSecurityAttemptCounter()*
0x60	*GetInfotypeValueData()*
0x61	*Dcm_SetProgConditions()*
0x62	*Dcm_GetProgConditions()*
0x65	*Indication()*
0x66	*Confirmation()*
0x67	*StartProtocol()*
0x79	*Dcm_SetDeauthenticatedRole()*
0xA1	*Dcm_TxConfirmation()*

Service ID	Service
0xA2	*Dcm_TriggerTransmit()*
0xA3	*Dcm_ProvideRecoveryStates()*
0xA4	*Dcm_OnRequestDetection()*
0xA6	*Dcm_GetTesterSourceAddress()*
0xA7	*Dcm_GetSecurityLevelFixedBytes()*
0xA8	*Dcm_ProcessVirtualRequest()*
0xA9	*Dcm_SetSecurityLevel()*
0xAA	*ReadData() (paged-data-reading)*
0xAB	*Dcm_GetRequestKind()*
0xAC	*Dcm_VsgSetSingle()*
0xAD	*Dcm_VsgIsActive()*
0xAE	*Dcm_VsgSetMultiple()*
0xAF	*Dcm_VsgIsActiveAnyOf()*
0xB0	*Dcm_HandleServiceExtern()*
0xB1	*Dcm_KeyMAsyncCertificateVerifyFinished()*
0xB2	*Dcm_CsmAsyncJobFinished()*
0xB3	*Dcm_SetSecurityBypass()*
0xB4	*Dcm_SetSpecificCauseCode()*
0xB5	*Dcm_CsmSecureCodingValidationFinished()*
0xB7	*Dcm_SetChannelReady()*
0xB8	*<Module>_<DiagnosticService>_<PreHandler>()*
0xB9	*<Module>_<DiagnosticService>_<PostHandler>()*
0xBA	*Dcm_DeauthenticateConnection()*
0xBB	*Dcm_AuthenticateConnection()*
0xBC	*Dcm_SetAuthenticationBypass()*
0xF0	DCM internal function

Table 2-7 DET Service IDs

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

Error Code		Description
0x01	DCM_E_INTERFACE_TIMEOUT	A timeout during interaction with another module occurs.
0x02	DCM_E_INTERFACE_RETURN_VALUE	The return value of called API is out of range.
0x03	DCM_E_INTERFACE_BUFFER_OVERFLOW	The boundary check of provided buffers fails.
0x05	DCM_E_UNINIT	Executing program code before the DCM is initialized.
0x06	DCM_E_PARAM	An API call with invalid parameter value.
0x07	DCM_E_PARAM_POINTER	An API call with invalid/null pointer parameter.
0x40	DCM_E_ILLEGAL_STATE	An internal DCM error, reaching an unexpected state.
0x41	DCM_E_INVALID_CONFIG	Marks an inconsistent configuration.
0x42	DCM_E_CRITICAL_ERROR	An invalid configuration detected at runtime. After this error code is reported, the DCM will deactivate itself and will not respond to any new diagnostic request.

Table 2-8 Errors reported to DET

2.5.2 Production Code Error Reporting

Production code related errors are not supported by DCM.

3 Integration

This chapter gives necessary information for the integration of the MICROSAR Classic DCM into an application environment of an ECU.

3.1 Embedded Implementation

The delivery of the DCM contains the files which are described in the chapters *3.1.1* and *3.1.2*:

3.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
Dcm.c	■		This is the implementation source file of the DCM (delivered only for the “pre-compile” variant).
Dcm_<Unit>.c	■		This is the implementation source file of the DCM <Unit> (delivered only for the “pre-compile” variant).
Dcm.h	■		This is the header file containing the APIs of DCM. **This is the only file that must be included by the application if an interaction with DCM is needed.**
Dcm_Cbk.h	■		This file contains all function prototypes of APIs called by other BSW-C (i.e. PduR, ComM, etc.).
Dcm_Types.h	■		This file contains all data types that shall be visible to the other components interacting with DCM.
Dcm_<Unit>.h Dcm_<Unit>Types.h	■		All these files belong to the DCM <Unit> part. **These files must not be included by any external source code.**
Dcm_bswmd.arxml	■		This file contains all DCM configuration parameters’ definitions.
Dcm_MemMap.arxml	■		This file contains all DCM specific memory sections.

Table 3-1 Static files

3.1.2 Dynamic Files

The dynamic files are generated by the DaVinci Configurator 5 generation tool.

File Name	Description
Dcm_Cfg.h	This file contains all pre-compile configuration settings of DCM (e.g. switches, constants, etc.).
Dcm_Lcfg.c	This file contains the link-time parameterization of DCM.
Dcm_Lcfg.h	This file contains all link-time parameters declarations and type definitions.
Dcm_PBcfg.c	This file contains all post-build loadable parameterization of DCM.
Dcm_PBcfg.h	This file contains all post-build loadable parameters declarations and type definitions.
Dcm_MemMap.h	This file contains the memory section mapping of DCM.

File Name	Description
Rte_Dcm.h	This file will be generated by the RTE.
Rte_Dcm_Type.h	This file will be generated by the RTE.
Dcm_swc.arxml	This AUTOSAR xml file is used for the configuration of the Rte. It contains the information to get prototypes of callback functions offered by other components.

Table 3-2 Generated files

3.2 Critical Sections

To protect internal data structures against modifications that will lead to data corruption, the DCM uses “Critical Sections” for blocking concurrent access, such as from lower transport layer and from the *Dcm_MainFunction()*.

The only method that DCM uses to handle the critical sections is:

> AUTOSAR Schedule Manager (SchM_Dcm.h is included)

Caution
You must take special care that the SchM implementing the critical section is already started before the DCM is run.

You must map the DCM critical sections to the appropriate resource locking method. DCM supports only the **`DCM_EXCLUSIVE_AREA_0`** and it shall be always mapped to **global interrupt disabling**, since DCM has always very short time critical sections. The real critical section duration depends on the performance of the controller used in your system, but the DCM critical section design restricts the code within to very few instructions and in very rear cases contains (internal) function calls, which usually are in-lined.

3.3 Compiler Abstraction and Memory Mapping

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

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

Compiler Abstraction Definitions / Memory Mapping Sections	DCM_CONST	DCM_CAL_PRM	DCM_CODE	DCM_VAR_NOINIT	DCM_VAR_INIT	DCM_APPL_CODE	DCM_APPL_DATA	DCM_CALLOUT_CODE	DCM_APPL_CONST	DCM_VAR_PBCFG	DCM_PBCFG
DCM_START_SEC_CONST_8 DCM_STOP_SEC_CONST_8	■										

DCM_START_SEC_CONST_16 DCM_STOP_SEC_CONST_16	■										
DCM_START_SEC_CONST_32 DCM_STOP_SEC_CONST_32	■										
DCM_START_SEC_CONST_UNSPECIFIED DCM_STOP_SEC_CONST_UNSPECIFIED	■										
DCM_START_SEC_CALIB_8 DCM_STOP_SEC_CALIB_8		■									
DCM_START_SEC_CALIB_16 DCM_STOP_SEC_CALIB_16		■									
DCM_START_SEC_CALIB_32 DCM_STOP_SEC_CALIB_32		■									
DCM_START_SEC_CALIB_UNSPECIFIED DCM_STOP_SEC_CALIB_UNSPECIFIED		■									
DCM_START_SEC_VAR_INIT_8 DCM_STOP_SEC_VAR_INIT_8					■						
DCM_START_SEC_VAR_NO_INIT_8 DCM_STOP_SEC_VAR_NO_INIT_8				■							
DCM_START_SEC_VAR_NO_INIT_16 DCM_STOP_SEC_VAR_NO_INIT_16				■							
DCM_START_SEC_VAR_INIT_32 DCM_STOP_SEC_VAR_INIT_32					■						
DCM_START_SEC_VAR_NO_INIT_32 DCM_STOP_SEC_VAR_NO_INIT_32				■							
DCM_START_SEC_VAR_NO_INIT_UNSPECIFIED DCM_STOP_SEC_VAR_NO_INIT_UNSPECIFIED				■							
DCM_START_SEC_CODE DCM_STOP_SEC_CODE			■								
DCM_START_SEC_CALLOUT_CODE DCM_STOP_SEC_CALLOUT_CODE								■			
DCM_START_SEC_APPL_CODE DCM_STOP_SEC_APPL_CODE						■					
DCM_START_SEC_VAR_PBCFG DCM_STOP_SEC_VAR_PBCFG										■	
DCM_START_SEC_PBCFG DCM_STOP_SEC_PBCFG											■

Table 3-3 Compiler abstraction and memory mapping

The compiler abstraction definitions of DCM_APPL_DATA and DCM_APPL_CONST refer to any RAM resp. ROM section defined by any external to DCM software module. This can be either BSW component or application data storage.

The DCM_APPL_CODE and DCM_CALLOUT_CODE definitions also refer to an external code section relative to DCM. These are memory locations, where the application code is placed. The difference between these two sections is that an application code in CALLOUT section is a DCM functionality extension (e.g. a complex device driver) and not a component in the matter of providing server application specific data or functionality (i.e. via RTE).

3.4 Considerations Using Request- and ResponseData Pointers in a Call-back

DCM is a half-duplex communication module and for memory usage optimization a single buffer is used for both request and response data. Therefore, if a call-back function contains both “ResponseData” and “RequestData” pointers, they may point to different addresses, but these are still memory locations within the same diagnostic buffer. So, if you start writing the response data, you probably will overwrite the request data. If the request data is still needed, while writing the response data, you must store it into temporary RAM location in your application software, before starting the write process.

For special handling in case of service RoutineControl (0x31) see chapter 4.24.5.

4 Diagnostic Service Implementation

The main goal of the DCM is to handle the diagnostic protocol services, defined by [10]. The only task the application has is to provide the required data, to write new data into its memory, access IO ports, etc. All these application tasks are ECU specific and have no dependency to the used diagnostic protocol.

The following chapters describe each diagnostic service that the DCM handles, including implementation and configuration aspects.

Each chapter provides tables that give an overview over the following information:

- Which implementation types of that diagnostic service are supported;
- If the service is internally handled, which subservices are supported and how they are or can be implemented.

For each of the about classifications the following implementation types are used:

- **Internal only** = by DCM
- **External only** = by application
- **Internal or external** = implemented by DCM, but can be overridden by application
- **Not allowed** = cannot be configured at all

Additionally, it is possible to configure a Pre- and Post-Handler callout function, which will be called right before or after the diagnostic service is processed. Please refer to section *8.47 How to Use Diagnostic Service Pre-/Post-Handler* for more information.

FAQ
If you miss a diagnostic service in the following chapter, it does only mean that the DCM does not provide any predefined implementation for it, but you can define it in DaVinci Configurator 5 and handle it within your application. If you try to specify an invalid service identifier, the DaVinci Configurator 5 will notify you about that and will deny the service definition.

FAQ
If not other stated every service that can be overridden by an application service handler may not have configured sub-services, but the application implementation of these services still can handle any by itself.

FAQ
Services can be temporarily disabled in DaVinci Configurator 5 without deleting the service entity from the project. This means, the service is still present in the ECUC file but is treated as if the container is not existent (dependencies are not validated and the container won't be considered during code generation). To disable a service set the parameter

/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSidTabUsed

to false.

4.1 RequestCurrentPowertrainDiagnosticData (0x01)

4.1.1 Functionality

This is a legislated OBD service that delivers some current values of ECU parameters.

4.1.2 Required Interfaces

- If service handled by DCM:
  - > *DataServices_<DataName>*
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.1.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-1 Service 0x01: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-2 Service 0x01: Supported subservices

This service is fully implemented by DCM.

4.1.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> All to be supported PIDs shall be defined within the following container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspPid

> For each PID to be supported by this service, the following parameter must be set to either SERVICE_01 or SERVICE 01_02:
/Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidService

> The data content of a PID shall be defined in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidData

> The access type to the data content of a PID can be defined in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidData/DcmDspPidService01

FAQ
There shall be no “availability ID” (i.e. 0x00, 0x20, 0x40 …, 0xE0) explicitly defined in the DCM configuration! All these IDs will be automatically calculated during the code generation process and supported by the DCM code.

FAQ
If any of the service’s PIDs shall be also readable by the corresponding UDS service (i.e. *ReadDataByIdentifier (0x22)* DIDs 0xF400 – 0xF4FF), the corresponding DIDs, **including the “availability DIDs”** shall be explicitly defined within the DCM configuration. This is required to support the optional read access condition checks on a DID operation.

Refer to *ReadDataByIdentifier (0x22)* for more details about OBD DID configuration particularities.

Note
For all PIDs implemented by the DEM, the according DEM APIs (e.g. Dem_DcmReadDataOfPID01) must be entered for the configuration parameter

Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidData/DcmDspPidService01/DcmDspPidDataReadFnc

4.2 RequestPowertrainFreezeFrameData (0x02)

4.2.1 Functionality

This is a legislated OBD service that delivers the contents of the OBD Freeze Frame, which consists of ECU parameter values stored by the fault memory module.

4.2.2 Required Interfaces

- If service handled by DCM:
  - > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.2.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-3 Service 0x02: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-4 Service 0x02: Supported subservices

This service is fully implemented by DCM.

4.2.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> All to be supported PIDs shall be defined within the following container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspPid

> For each PID to be supported by this service, the following parameter must be set to either SERVICE_02 or SERVICE 01_02:
/Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidService

FAQ
There shall be no “availability ID” (i.e. 0x00, 0x20, 0x40 …, 0xE0) explicitly defined in the DCM configuration! All these IDs will be automatically calculated during the code generation process and supported by the DCM code.

4.3 RequestEmissionRelatedDTC (0x03)

4.3.1 Functionality

This is a legislated OBD service that delivers all DTCs with status “confirmed”.

4.3.2 Required Interfaces

- If service handled by DCM:
  > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.3.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
SubServiceID				■

Table 4-5 Service 0x03: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-6 Service 0x03: Supported subservices

This service is fully implemented by DCM.

4.3.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

4.4 ClearEmissionRelatedDTC (0x04)

4.4.1 Functionality

This is a legislated OBD service that clears all emission related DTCs.

4.4.2 Required Interfaces

- If service handled by DCM:
  > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.4.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-7 Service 0x04: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-8 Service 0x04: Supported subservices

This service is fully implemented by DCM.

4.4.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

4.5 RequestOnBoardMonitorTestResults (0x06)

4.5.1 Functionality

This is a legislated OBD service that delivers monitor specific test results.

4.5.2 Required Interfaces

- If service handled by DCM:
  - > *DtrServices* (if no OBD DTR support by DEM)
  - > Refer to chapter *5.3 Services used by DCM* for the DEM component (if OBD DTR is supported by DEM).
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.5.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-9 Service 0x06: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-10 Service 0x06: Supported subservices

This service is fully implemented by DCM.

Caution
Depending on the DEM SWS AR version and setup, the OBDMID configuration and data handling is either implemented by DCM or DEM.

Please refer to the configuration aspects in the following chapters for more details:

> *4.5.4 Configuration Aspects*
> *8.30 How to Switch Between OBD DTR Support by DCM and DEM*

4.5.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> If the OBDMID configuration and data handling is to be supported by DEM, the following parameters will not be required, resp. will be ignored during the DCM configuration code generation.

  > All to be supported MIDs shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspTestResultByObdmid/DcmDspTestResultObdmidTid

  > For each MID to be supported by this service, the corresponding TIDs shall be associated: /Dcm/DcmConfigSet/DcmDsp/DcmDspTestResultByObdmid/DcmDspTestResultObdmidTid/DcmDspTestResultObdmidTids

  > The access type to the data content of a MIDTID can be defined in the container: /Dcm/DcmConfigSet/DcmDsp/DcmDspTestResultByObdmid/DcmDspTestResultObdmidTid/DcmDspTestResultObdmidTids/DcmDspTestResultObdmidTidUsePort

FAQ
There shall be no “availability ID” (i.e. 0x00, 0x20, 0x40 …, 0xE0) explicitly defined in the DCM configuration! All these IDs will be automatically calculated during the code generation process and supported by the DCM code.

FAQ
If any of the service’s MIDs shall be also readable by the corresponding UDS service (i.e. *ReadDataByIdentifier (0x22)* DIDs 0xF600 – 0xF6FF), the corresponding DIDs, **including the “availability DIDs”** shall be explicitly defined within the DCM configuration. This is required to support the optional read access condition checks on a DID operation.

Refer to *ReadDataByIdentifier (0x22)* for more details about OBD DID configuration particularities.

4.6 RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle (0x07)

4.6.1 Functionality

This is a legislated OBD service that delivers all DTCs with status “pending”.

4.6.2 Required Interfaces

- If service handled by DCM:
  - > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.6.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-11 Service 0x07: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-12 Service 0x07: Supported subservices

This service is fully implemented by DCM.

4.6.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

4.7 RequestControlOfOnBoardSystemTestOrComponent (0x08)

4.7.1 Functionality

This is a legislated OBD service that starts a routine within the ECU.

4.7.2 Required Interfaces

- If service handled by DCM:
  > *RequestControlServices_<TIDName>*
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.7.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-13 Service 0x08: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-14 Service 0x08: Supported subservices

This service is fully implemented by DCM.

4.7.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> Each to be supported TIDs shall be defined in a container: /Dcm/DcmConfigSet/DcmDsp/DcmDspRequestControl

> The request data content size of a TID shall be defined in the container: /Dcm/DcmConfigSet/DcmDsp/DcmDspRequestControl/DcmDspRequestControlInBufferSize

> The response data content size of a TID shall be defined in the container: /Dcm/DcmConfigSet/DcmDsp/DcmDspRequestControl/DcmDspRequestControlOutBufferSize

> The access type to the data content of a PID can be defined in the container: /Dcm/DcmConfigSet/DcmDsp/DcmDspRequestControl/DcmDspRequestControlUsePort

FAQ
There shall be no “availability ID” (i.e. 0x00, 0x20, 0x40 …, 0xE0) explicitly defined in the DCM configuration! All these IDs will be automatically calculated during the code generation process and supported by the DCM code.

FAQ
If any of the service’s PIDs shall be also readable by the corresponding UDS service (i.e. *RoutineControl (0x31)* DIDs 0xE000 – 0xE0FF), the corresponding RIDs, **including the “availability RIDs”** shall be explicitly defined within the DCM configuration. This is required to support the optional control access condition checks on a RID.

Refer to *RoutineControl (0x31)* for more details about OBD RID configuration particularities.

4.8 RequestVehicleInformation (0x09)

4.8.1 Functionality

This is a legislated OBD service that delivers some vehicle identification information.

4.8.2 Required Interfaces

- If service handled by DCM:
  > *InfotypeServices_<VEHINFODATA>*
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.8.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-15 Service 0x09: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-16 Service 0x09: Supported subservices

This service is fully implemented by DCM.

4.8.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> Each to be supported VID shall be defined in a container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspVehInfo

FAQ
There shall be no “availability ID” (i.e. 0x00, 0x20, 0x40 …, 0xE0) explicitly defined in the DCM configuration! All these IDs will be automatically calculated during the code generation process and supported by the DCM code.

FAQ
If any of the service’s VIDs shall be also readable by the corresponding UDS service (i.e. *ReadDataByIdentifier (0x22)* DIDs 0xF800 – 0xF8FF), the corresponding DIDs, **including the “availability DIDs”** shall be explicitly defined within the DCM configuration. This is required to support the optional read access condition checks on a DID operation.

Refer to *ReadDataByIdentifier (0x22)* for more details about OBD DID configuration particularities.

> The data content of a VID shall be defined in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspVehInfo/DcmDspVehInfoData

FAQ
In case the OBD VID data length shall be variable, the configuration parameter /Dcm/DcmConfigSet/DcmDsp/DcmDspVehInfo/DcmDspVehInfoData/DcmDspVehInfoDataSize will specify the maximum data size of the VID. This value will be passed as an input to the API *GetInfotypeValueData()*.

> The access type to the data content of a VID can be defined in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspVehInfo/DcmDspVehInfoData/DcmDspVehInfoDataUsePort

4.9 RequestEmissionRelatedDTCsWithPermanentStatus (0x0A)

4.9.1 Functionality

This is a legislated OBD service that delivers all DTCs with status “permanent”.

4.9.2 Required Interfaces

- If service handled by DCM:
  - > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.9.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-17 Service 0x0A: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-18 Service 0x0A: Supported subservices

This service is fully implemented by DCM.

4.9.4 Configuration Aspects

- > This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService
- > No sub-functions shall be defined within the above defined service container

4.10 DiagnosticSessionControl (0x10)

4.10.1 Functionality

This service manages the diagnostic session state in the ECU.

4.10.2 Required Interfaces

- > *DcmDiagnosticSessionControl*

4.10.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID	■			
SubServiceID	■			

Table 4-19 Service 0x10: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00				■
0x01	■			
0x02	■			
0x03	■			
0x04 … 0x7E	■			
0x7F … 0xFF				■

Table 4-20 Service 0x10: Supported subservices

This service is fully implemented by DCM.

4.10.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

Caution
This service is mandatory and therefore may not be missing in the configuration and cannot be overridden by an application implementation.

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

> For each defined sub-function there shall be a corresponding session level defined: /Dcm/DcmConfigSet/DcmDsp/DcmDspSession

For each session, there must be also defined the P2/P2Start timings: /Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionRow/DcmDspSessionP2ServerMax and /Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionRow/DcmDspSessionP2StarServerMax

FAQ
The P2/P2Start timings above will be reported to the diagnostic client within the positive response of this service. These timings will apply if the DCM is in the corresponding session. DCM is designed to send the RCR-RP not later than the configured P2/P2Star time. Depending on the project integration specifics and main-functions scheduling of the communication stack (interfaces, transport layers, etc.) it may lead to a delayed RCR-RP responses and failing compliance tests. Still, you have the opportunity to adjust the DCM internal timer values by specifying a diagnostic protocol specific (i.e. UDS and OBD may have different adjustments) timing corrections. Please refer to the following parameters in the DCM configuration: /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmTimStrP2ServerAdjust and /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmTimStrP2StarServerAdjust

4.11 EcuReset (0x11)

4.11.1 Functionality

This service implementation provides the reset functionality within the ECU.

Note
Once one of the following reset modes: HardReset, SoftReset and KeyOnOffReset is being requested, after sending the positive response resp. finishing service processing without positive response, DCM will not accept any further diagnostic request until the ECU is reset or *Dcm_ResetToDefaultSession()* is called.

FAQ
In some cases it is required not to perform a real reset of the ECU, but only to switch into the default session and reset all active diagnostic jobs. If this kind of reset implementation is required, then the application shall just call the *Dcm_ResetToDefaultSession()* provided port operation once the Mode_Switch operation for the *DcmEcuReset* mode declaration group is triggered.

4.11.2 Required Interfaces

- If service handled by DCM:
  - *DcmEcuReset*
  - *DcmModeRapidPowerShutDown*
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

4.11.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID		■		

Table 4-21 Service 0x11: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00				■
0x01		■		
0x02		■		
0x03		■		
0x04		■		
0x05		■		
0x06 … 0x7E			■	
0x7F … 0xFF				■

Table 4-22 Service 0x11: Supported subservices

All in *Table 4-22 Service 0x11: Supported subservices* sub-functions marked as internally handled by DCM are fully implemented and no application interaction is necessary.

Caution
If any of the service's sub-functions 0x01-0x05 are implemented externally (user defined implementation), the corresponding mode switches (if required) shall be triggered by the user implementation.

The mode declaration groups (*DcmEcuReset* and *DcmModeRapidPowerShutDown*) will exist only if at least one of the corresponding sub-functions is still handled by DCM.

4.11.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

> For internally supported sub-functions 0x01, 0x02 and 0x03 it can be configured individually whether the positive response for the reset action shall be send before or after the execution.

  > Settings for each supported sub-function can be defined as a container in the table: /Dcm/DcmConfigSet/DcmDsp/DcmDspEcuReset

> If sub-function 0x04 is to be supported, additionally the following parameter shall be configured: /Dcm/DcmConfigSet/DcmDsp/DcmDspPowerDownTime

4.12 ClearDiagnosticInformation (0x14)

4.12.1 Functionality

This service clears the stored fault memory content.

4.12.2 Required Interfaces

- If service handled by DCM:
  > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.12.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-23 Service 0x14: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-24 Service 0x14: Supported subservices

This service is fully implemented by DCM.

4.12.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> If UserDefined Memory selection for service 0x14 shall be supported, a single container of the following type has to be defined and configured accordingly: /Dcm/DcmConfigSet/DcmDsp/DcmDspClearDTC

  > Select whether the UserDefined Memory selection shall be enabled or disabled in the following parameter: /Dcm/DcmConfigSet/DcmDsp/DcmDspClearDTC/DcmDspClearDTCMemorySelectionEnabled

  > Specify the supported user memory ID in the following parameter. If no memory identifier is specified, the Dcm will accept any value in the request. Otherwise, the Dcm will accept only the user memory ID that is configured here: /Dcm/DcmConfigSet/DcmDsp/DcmDspClearDTC/DcmDspClearDTCMemoryUserMemoryId

  > If an additional authentication check for the specified UserDefined memory ID shall be supported, the allowed roles can be referenced here: /Dcm/DcmConfigSet/DcmDsp/DcmDspClearDTC/DcmDspClearDTCInformationUserDefinedMemoryRoleRef

4.13 ReadDTCInformation (0x19)

4.13.1 Functionality

This service reads the stored fault memory information using the DEM data access API.

4.13.2 Required Interfaces

- If service handled by DCM:
  > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.13.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID		■		

Table 4-25 Service 0x19: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00				■
0x01 … 0x1A		■		

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x1B … 0x41			■	
0x42		■		
0x43 … 0x54			■	
0x55		■		
0x56		■		
0x57 … 0x7E			■	
0x7F … 0xFF				■

Table 4-26 Service 0x19: Supported subservices

All above sub-functions marked as internally handled by DCM are fully implemented and no application interaction is necessary.

FAQ
All only to WWH-OBD related sub-functions (e.g. 0x42) require a valid WWH-OBD license to be handled internally in DCM. Otherwise they must be implemented within an external CDD module.

FAQ
Most only to OBD related sub-functions (e.g. 0x56) require a valid OBD license to be handled internally in DCM. Otherwise they must be implemented within an external CDD module.

FAQ
Sub-functions 0x16, 0x1A and 0x56 can be handled internally by DCM only together with a MICROSAR Classic DEM. If sub-functions 0x16, 0x1A or 0x56 are required with a non MICROSAR Classic DEM, the sub-functions must be implemented externally.

4.13.3.1 Reporting Stored DTC Environment Data

Note
If this DCM module is used together with either a MICROSAR Classic DEM in any version, or a non-MICROSAR Classic DEM with DemApiVersion DCM_DEM_API_4_03_00 or newer, it is not necessary to use or change this configuration. DCM will automatically take the DEM settings regarding the supported record numbers.

For all snapshot and extended data record sub-functions, DCM module requires additional input from the ECU configuration. To be able to report properly all related record numbers when the records masks 0xFF or 0xFE are requested, the DCM configuration has been extended by a new parameter hierarchy:

/Dcm/DcmConfigSet/DcmDsp/DcmDspFaultMemory/DcmDspFaultMemoryRecords.

These new parameters allow DEM configuration independent parameterization of DCM.

More details about them follow in next chapter and in the online help of each parameter under this container.

4.13.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

FAQ
For all user defined sub-functions (marked as “external only” in *Table 4-26 Service 0x19: Supported subservices*) the sub-function specific request length check shall be performed by the corresponding sub-function processor implementation. This may lead to a deviation of the defined in [10] NRC prioritization on a double error (i.e. wrong security access level and invalid sub-function length). Currently this is unavoidable since [1] does not provide a request length configuration option on sub-service level.

FAQ
For sub-functions 0x16, 0x1A, 0x55 and 0x56 DemApiVersion must be at least DCM_DEM_API_4_03_00.

> If one of the sub-functions 0x17-0x19 shall be supported, a MemoryIdentifier is optionally possible to be specified:
/Dcm/DcmConfigSet/DcmDsp/DcmDspFaultMemory/DcmDspFaultMemoryUserMemoryIdInfo/DcmDspFaultMemoryUserMemoryId.
If an additional authentication check for the specified MemoryIdentifer shall be supported, the allowed roles can be referenced here:
/Dcm/DcmConfigSet/DcmDsp/DcmDspFaultMemory/DcmDspFaultMemoryUserMemor

yIdInfo/DcmDspReadDTCInformationUserDefinedFaultMemoryRoleRef.
For more details please refer to the parameter's online help within the configuration tool.

> If a non-MICROSAR Classic DEM with DemApiVersion earlier than DCM_DEM_API_4_03_00 is used together with DCM and one of the stored DTC environment data reporting sub-functions of this diagnostic service is to be supported, all related record ranges shall be specified in the ECUC under the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspFaultMemory/DcmDspFaultMemoryRecords

4.14 ReadDataByIdentifier (0x22)

4.14.1 Functionality

This service provides read access to data structures within the ECU, marked by an identifier (DID).

The tester may simultaneously access multiple DIDs in a single request. The maximum allowed DID list length is configurable (refer to *4.14.4 Configuration Aspects* for more details).

If SilentBSW is enabled (see chapter *8.35 Usage Hints for Operation with SilentBSW*) and paged reading is disabled (see *8.24 How to Save RAM using Paged-Buffer for Large DIDs*), when reading a DID with variable length, the DCM verifies that the maximum configured DID length does not exceed the internal buffer. Therefore, DIDs with variable lengths can only be read if, after reading all DIDs preceeding the DID with variable length, the remaining space in the buffer is at least as large as the maximum configured size of the DID with variable length. The size checks for succeeding DIDs is not affected by that.

4.14.2 Required Interfaces

- If service handled by DCM:
  - > *DataServices_<DataName>*
  - > *DataServices_DIDRange_<RangeName>*
  - > *DataServices*
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.14.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-27 Service 0x22: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
All				■

Table 4-28 Service 0x22: Supported subservices

The protocol handling of this service is fully implemented by DCM. The data reported by each DID will be provided by the application via service calls or call outs.

Caution
If you intend to use DID ranges, please read carefully chapter *8.20 Handling with DID Ranges* to learn important particularities.

FAQ
In case very large DID data must be carried out from the application an optimized data reading process can be used to save RAM. For details please refer to 8.24 How to Save RAM using Paged-Buffer for Large DIDs.

4.14.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container;

> All to be supported readable DIDs shall be defined within the following container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDid

> The read operation over a DID is defined by:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidRead

> The maximum number of simultaneously requested DIDs shall be defined by:
/Dcm/DcmConfigSet/DcmDsp/DcmDspMaxDidToRead

> For each DID data signal the corresponding container shall be configured:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData

> The check condition read operation is optional and if not used can be deactivated via:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataConditionCheckReadFncUsed

> For NvRam signal access select the value USE_BLOCK_ID in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort

> A NvRam block Id must be referenced:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataBlockIdRef

FAQ

Particularities for OBD DIDs (i.e. all within [0xF400-0xF8FF]):

> If DEM handles DTR values, please consider also chapter *8.30 How to Switch Between OBD DTR Support by DCM and DEM* for information on the DIDs.

> Any OBD availability DID (e.g. 0xF400, 0xF420, 0xF600, 0xF620, 0xF880, 0xF8E0, etc.) will always be implemented by DCM. They will return the corresponding DID availability mask value as described in [12].

> Every DID in the OBD range that covers a corresponding OBD PID, MID or VID, shall not contain any data definition. The concrete data will be read out by DCM directly using the corresponding OBD service data access method. For such DIDs, there also will be no RTE DataServices port or callback generated.

> Any OBD DID, that neither is an availability DID, nor covers any existing OBD PID, MID or VID, will be handled as a generic DID and shall be configured regularly.

Caution

Limitations for OBDonUDS DIDs in range [0xF700-0xF7FF]:

According to [10], the DID range 0xF700-0xF7FF was reserved for OBDMonitorDataIdentifier to represent future defined OBD/EOBD on-board monitoring result values. This was changed with [11], so that this DID range is now reserved for OBDDataIdentifier for regulated emissions-related data.

According to [15], service 0x22 makes use of the DID ranges [0xF400-0xF5FF] and [0xF700-0xF7FF]. The supported DID range shall be a chain, which means that DID 0xF4E0 indicates about the support of DID 0xF500. In same way, DID 0xF5E0 shall indicate support of DID 0xF700. Currently, DID 0xF5E0 is not aware about any DID supported in the range [0xF700-0xF7FF].

4.15 ReadMemoryByAddress (0x23)

4.15.1 Functionality

This service provides direct read access to the physical memory of the ECU. All readable memory areas and their access preconditions are to be configured as documented in *4.15.4 Configuration Aspects*.

4.15.2 Required Interfaces

- If service handled by DCM:
  > *Dcm_ReadMemory()*
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.15.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-29 Service 0x23: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-30 Service 0x23: Supported subservices

The protocol handling of this service is fully implemented by DCM. This includes:

- Validating and evaluating the ALFID byte
- Parsing the requested memory address and size parameters
- Validating the requested memory block against the DCM memory configuration:
  - Supported requested memory area by the ECU
  - Memory area access preconditions (e.g. security access, mode rules)

The memory access will then be provided by the application via a call out.

FAQ
All readable memory ranges will be considered during the definition of a DDID with *DynamicallyDefineDataIdentifier (0x2C)*.

4.15.4 Configuration Aspects

- This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService
- No sub-functions shall be defined within the above defined service container.
- All to be supported readable memory ranges shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspMemory

  They can be configured either using numerical or symbolic representations.

> For numerical representation:
/Dcm/DcmConfigSet/DcmDsp/DcmDspMemory/DcmDspMemoryIdInfos/DcmDspMemoryIdInfo/DcmDspReadMemoryRangeInfos

> For symbolic representation:
/Dcm/DcmConfigSet/DcmDsp/DcmDspMemory/DcmDspMemoryIdInfos/DcmDspMemoryIdInfo/DcmDspReadMemoryRangeByLabelInfos

The header file, containing the definition of the labels, can be added by
/Dcm/DcmConfigSet/DcmGeneral/DcmHeaderFileInclusion

Caution
Memory ranges represented by symbols (DcmDspReadMemoryRangeByLabelLow resp. DcmDspReadMemoryRangeByLabelHigh) shall not overlap each other and have correct relationships between start and end addresses (start address <= end address).

Note
It can be decided individually per memory identifier (DcmDspMemoryIdInfo) whether the memory ranges are represented numerically or symbolically.

4.16 ReadScalingDataByIdentifier (0x24)

4.16.1 Functionality

This service provides read access to scaling information of each data within a DID.

4.16.2 Required Interfaces

- If service handled by DCM:
  > *DataServices_<DataName>*
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.16.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-31 Service 0x24: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-32 Service 0x24: Supported subservices

The protocol handling of this service is fully implemented by DCM. The data reported by each DID will be provided by the application via service calls or call outs.

FAQ
AUTOSAR does not provide a means for specifying session, security or mode rule restrictions on scaling information operation per DID. Thus, the only way to limit the access to the scaling data is by limiting the access to the whole service 0x24 under the corresponding parameter (e.g. DcmDsdSidTabSecurityLevelRef) in /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

4.16.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> All to be supported scaling DIDs shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspDid

> For each DID data signal the corresponding container shall be configured: /Dcm/DcmConfigSet/DcmDsp/DcmDspData

> For each DID data signal the corresponding container shall be configured in its scaling size: /Dcm/DcmConfigSet/DcmDsp/DcmDspDataInfo/DcmDspDataScalingInfoSize

4.17 SecurityAccess (0x27)

4.17.1 Functionality

This service manages the security level of the ECU used to constrain the diagnostic access to critical services like writing data in restricted areas.

4.17.2 Required Interfaces

The following interfaces must be available when service 0x27 is used:

- If service handled by DCM:
  - *SecurityAccess_<SecurityLevelName>*
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*
  - *Dcm_SetSecurityLevel()*

4.17.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID	■			

Table 4-33 Service 0x27: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00				■
0x01 … 0x7D	■			
0x7E … 0xFF				■

Table 4-34 Service 0x27: Supported subservices

By default, this service is fully implemented by DCM. If the internal implementation is used, the following specifics must be considered:

If the ECU shall support “failed attempt monitoring”, it can be chosen between two strategies on how to avoid brute-force-attack bypass via ECU reset.

> Dynamic power-on delay time management:

The attempt counter shall be stored by the application (e.g. into a NvM block), so at next ECU power on/reset event its value can be recovered.

Note
According to [10], after power up/reset of the ECU, the delay timer shall be started if the attempt counter is not zero. The implementation of the DCM only starts the delay timer if the configured attempt counter limit is exceeded.

To reproduce the behavior specified in [10], the application shall always write the attempt counter limit (or 255, which is configuration independent) to the NvM in the *SetSecurityAttemptCounter()* callout if the passed value is not zero.

> Static power-on delay management:

The attempt counter will not be stored into the NvM (by the application), but instead DCM will use the “delay time on boot” setting to insert a penalty time at each power on cycle, regardless of the last attempt counter state. This means that even if during the last power-on cycle there was no failed attempt, the ECU will not accept any request for service 0x27 for that level, having set up “delay time on power on”.

Please, refer the configuration related chapter below to find the corresponding DCM settings that affect the brute-force-attack bypass strategy.

MICROSAR Classic DCM provides an optional extension of the security access level configuration if some fixed bytes for the seed/key value calculation are needed. For details, please refer to chapter *8.25 How to Get Security-Access Level Specific Fixed Byte Values*.

4.17.4 Configuration Aspects

- This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService
- All to be supported sub-functions shall be defined within the above defined service container as sub-service containers: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService
- There shall always be a pair of sub-functions per security level (e.g. 0x01 for “get seed” and 0x02 for the corresponding “send key” sub-function).
- For each pair there shall always be a corresponding security level defined: /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow
- If a notification on a security access level state change is required, the option described in *8.18 How to Know When the Security Access Level Changes* shall be enabled.
- Specify whether a single (shared among all security levels) or multiple (per security level) instances of the attempt counter shall be supported: /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecuritySingleInstanceAttemptMonitor
- Specify whether a single (shared among all security levels) or multiple (per security level) instances of the delay timer shall be supported: /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecuritySingleInstanceDelayTimer
- Specify whether a non-volatile storage of the attempt counter is required for a certain level: /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow/DcmDspSecurityAttemptCounterEnabled
- Specify whether an unconditional delay timer start is required for a certain level: /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow/DcmDspSecurityDelayTimeOnBoot

FAQ
You can only choose to have either DcmDspSecurityAttemptCounterEnabled or DcmDspSecurityDelayTimeOnBoot. Both settings cannot be combined.

- The access type to the security level specific operations can be defined using the following parameter:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow/DcmDspSecurityUsePort

> Specify the attempt counter/timer recovery replacement strategy, in case the last stored attempt counter value is no more readable:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow/DcmDspSecurityDelayTimeOnFailedGetAttemptCounter

> Specify whether the attempt counters shall be reset when the delay timer expires:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityResetAttemptCounterOnTimeout

Post-build selectable

In addition to the post-build support described in chapter *8.19.1.2 Diagnostic Services Part* the seed, key, and access data record size parameters can be configured with variant specific values:

> DcmDspSecuritySeedSize

> DcmDspSecurityKeySize

> DcmDspSecurityADRSize

The provided ports for each security level (see *5.6.1.2.3 SecurityAccess_<SecurityLevelName>*) remain invariant. Each port operation (*GetSeed() (with SADR), GetSeed() (without SADR), CompareKey()*) specifies always the maximum size over all variants of the corresponding parameter. Nevertheless, Dcm will execute variant specific request length checks and will respond with the configured seed length of the active variant.

Caution
Depending on the configuration (e.g., active implicit RTE communication), the RTE might intermediately buffer data. Because the SWC description is invariant, the RTE will therefore read/write the maximum parameter length, instead of the variant specific value. This must be considered by the application in its buffer management. However, the application shall evaluate only the relevant number of bytes for the active variant.

4.18 CommunicationControl (0x28)

4.18.1 Functionality

This service manages the communication state of both reception and transmission path of the ECU.

4.18.2 Required Interfaces

- If service handled by DCM:

  > Refer to chapter *5.3 Services used by DCM* for the BswM component.

- If service handled by the application:

> *<Module>_<DiagnosticService>()*

4.18.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID		■		

Table 4-35 Service 0x28: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00 … 0x03	■			
0x04 … 0x05			■	
0x06 … 0x3F				■
0x40 … 0x7E			■	
0x7F … 0xFF				■

Table 4-36 Service 0x28: Supported subservices

This service is fully implemented by DCM with the following limitations:

For the sub-network id parameter only, the values “CurrentSubNetwork” and “AllSubNetworks” are supported. The third type: “SpecificSubNetworkId” is currently not supported.

4.18.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

FAQ
For all user defined sub-functions (marked as “external only” in *Table 4-36 Service 0x28: Supported subservices*) the sub-function specific request length check shall be performed by the corresponding sub-function processor implementation. This may lead to a deviation of the defined in [10] NRC prioritization on a double error (i.e. wrong security access level and invalid sub-function length). Currently this is unavoidable since [1] does not provide a request length configuration option on sub-service level.

> All other for this service relevant properties shall be configured under: /Dcm/DcmConfigSet/DcmDsp/DcmDspComControl

FAQ
It is important that if UDS parameter “CommunicationType” 0x0X (AllNetworks) shall be supported by DCM, that the corresponding channels are configured appropriately under the following configuration containers: /Dcm/DcmConfigSet/DcmDsp/DcmDspComControl/DcmDspComControlAllChannel

> In case DCM shall monitor any critical condition under which this service shall no longer be active, put a reference to that condition using parameter: /Dcm/DcmConfigSet/DcmDsp/DcmDspComControl/DcmDspComControlSetting/DcmDspComControlCommunicationReEnableModeRuleRef

4.19 Authentication (0x29)

4.19.1 Functionality

This service manages the authentication states of connections to the ECU used to provide access to diagnostic services based on roles and white lists contained in the authentication certificates. Please refer also to section *8.41* for further information.

4.19.2 Required Interfaces

The following interfaces must be available when service 0x29 is used internally:

> Refer to chapter *5.3 Services used by DCM* for the Csm and KeyM components.

If user specific authentication role handling needs to be supported:

> *Dcm_GetAuthenticationRoles()*

> *Dcm_SetDeauthenticatedRole()*

When service 0x29 is implemented completely externally and the DCM inherent state checks shall be used:

> *Dcm_DeauthenticateConnection()*

> *Dcm_AuthenticateConnection()*

> *Dcm_ConnectionAuthenticated()*

4.19.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID		■		

Table 4-37 Service 0x29: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00 … 0x04		■		
0x05 … 0x07			■	
0x08		■		
0x09 … 0xFF			■	

Table 4-38 Service 0x29: Supported subservices

Caution

The authentication states are handled internally by the Dcm.

The DCM provides APIs to set the authentication states only when service 0x29 is implemented fully externally. Thus, if the DCM inherent state checks are intended to be used, service 0x29 has to be implemented fully externally.

If any subservice is configured to be implemented internally the APIs to set the authentication states are not available. Thus, the authentication states to provide access to diagnostic services cannot be set by the application. In this case the access to diagnostic services might not work as intended.

Regardless of the implementation (internal/external), the API to deauthenticate a tester is always available. This allows the application to deauthenticate the tester on demand, if further checks, e.g. certificate expiration, enforce this.

Note

If one of the subservices 0x00 - 0x03 or 0x08 is implemented externally, all the other subservices from the range 0x00 - 0x03 and 0x08 must be implemented externally as well. Therefore, only the following configuration options are available for the subservices 0x00 - 0x03 and 0x08:

> all subservices in the range 0x00 - 0x03 and 0x08 implemented internally

> all subservices in the range 0x00 - 0x03 and 0x08 implemented externally

Note

If the subservice 0x00 and 0x08 are implemented externally, the length check is performed by the Dcm. Therefore, the length of the request must be always 2 (service ID and subservice ID). If the length of the request differs from 2, the Dcm sends a negative response with NRC 0x13 (incorrectMessageLengthOrInvalidFormat).

Note

Service 0x29 is internally not supported, when generic connections (see also section *8.39*) are configured. Nevertheless, in a configuration in which both non-generic and generic connections exist, it is possible to configure service 0x29 for the non-generic connections (see also section *8.12.4*). It is also possible to configure service 0x29 with generic connections, as long as the service is implemented externally.

The APIs *Dcm_DeauthenticateConnection()* and *Dcm_AuthenticateConnection()* only differentiate between different connectionIds but not different TesterSourceAdresses in case of generic connections. So, if these APIs are used with generic connections, the application has to take care to e.g. deauthenticate a connection with a different TesterSourceAdress before processing a request.

Note

When Service 0x29 is implemented completely externally but the DCM inherent state checks shall be used, the DCM inherent Authentication Default Timer (DcmDspAuthenticationDefaultSessionTimeOut) is not supported. Additionally, the DCM will not automatically deauthenticate a tester in case of a S3 timeout.

Note

When Service 0x29 is implemented completely externally but the DCM inherent state checks shall be used, the DCM will not take care of persistence of authentication states. Nevertheless, the application can restore these states after the initialization by calling the API *Dcm_AuthenticateConnection()*.

4.19.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

> Global configurations for Authentication shall be configured in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication

Note
With the configuration of the authentication max size parameters, the DCM generator is able to validate that the request and response will fit into the configured main buffer.
Otherwise, there could be the case that a request message or a response is too long and cannot be processed because the configured buffer is too small.

> If the DCM inherent state checks shall be used, all connections, which shall be able to be authenticated, shall be referenced by the following container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationConnection/DcmDspAuthenticationConnectionMainConnectionRef

Note
If service 0x29 is implemented completely externally and no connection is referenced, *Dcm_DeauthenticateConnection()* and *Dcm_AuthenticateConnection()* are not available.

> The connection specific Csm certificate and KeyM Job references shall be configured in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationConnections

Note
If a White List Element Ref parameter is not referenced, then that particular white list is disabled for this connection.

> All connections to be supported by this service shall be configured to reference specific configured Crypto Service Manager [9] jobs defined in the container:
/Csm/CsmJobs/CsmJob

  > The CsmJob needs to be configured as asynchronous:
/Csm/CsmJobs/CsmJob/CsmProcessingMode

  > The CsmJob must reference a CsmPrimitive (CsmRandomGenerate, CsmSignatureGenerate and CsmSignatureVerify) depending on its functionality:
/Csm/CsmJobs/CsmJob/CsmJobPrimitiveRef

  > The CsmJob must reference a primitive callback since it is configured asynchronous:
/Csm/CsmJobs/CsmJob/CsmJobPrimitiveCallbackRef

- The callback notification function *Dcm_CsmAsyncJobFinished()* shall be configured for all used Csm jobs (CsmRandomGenerate, CsmSignatureGenerate and CsmSignatureVerify) in: /Csm/CsmCallbacks/CsmCallback/CsmCallbackFunc
- If a MICROSAR Classic Csm is used, version CSM_ASR_VERSION_R19_11 must be selected for: /Csm/CsmCallbacks/CsmCallback/CsmBswCallbackAsrVersion

> All connections to be supported by this service shall be configured to reference certificates defined in the container: /KeyM/KeyMCertificates

> The callback notification function *Dcm_KeyMAsyncCertificateVerifyFinished()* shall be configured in: /KeyM/KeyMCertificate/KeyMCertificateVerifyCallbackNotificationFunc

> All connections to be supported by this service shall be configured to access an authentication role element within a certificate defined in the container: /KeyM/KeyMCertificates/[Certificate]/KeyMCertificateElements

> Connection configurations may reference white list certificate elements for DIDs, Memory Selection, RIDs, and Services identically to the container above: /KeyM/KeyMCertificates/[Certificate]/KeyMCertificateElements

> For sub-function 0x04 (transmitCertificate) it is required to configure the certificate references in the container: /Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationTransmitCertificate

> For detailed information about the configuration of the components Csm and KeyM, see [20] and [21].

Note
For service tables which contain OBD and UDS services including service 0x29, all connections which are using such a service table need to be configured to support authentication. Independent of if that connection will only use OBD services and does not use authentication.

> An API for setting a connection specific deauthenticated role can be enabled here: Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationSetDeauthenticatedRoleApi

> When DCM is processing a sub-function 0x03 request, the role information is directly read from the certificate via KeyM_CertElementGet(). DCM expects the authentication role information in the certificate to be encoded according to the AUTOSAR format [3]. However, in some ECU projects it may be necessary to use certificates in which the format of the authentication roles is user-defined.

  In such use cases, DCM can be configured to no longer call KeyM_CertElementGet() to read the role information, but rather the callout function *Dcm_GetAuthenticationRoles()*. In this function, the authentication role information can be retrieved from KeyM directly by the application and converted to the AUTOSAR format before they are returned to DCM.

If the user specific authentication roles are to be supported, the following parameter shall be enabled:
/Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationUserSpecificRoles

4.20 ReadDataByPeriodicIdentifier (0x2A)

4.20.1 Functionality

This service provides read access to data structures within the ECU, marked by a periodic identifier (PDID). These are all DIDs in range [0xF200 – 0xF2FF].

The tester may schedule multiple PDIDs in a single request. The maximum allowed PDID list length is configurable (refer to *4.20.4 Configuration Aspects*).

Optionally, DCM is able to stop automatically the periodic transmission of any scheduled PDID that cannot be accessed any more, after a diagnostic session/security access level change. Refer to *4.20.4 Configuration Aspects* for details about this setting.

DCM is also able to induce a delay between periodic messages to reduce bus load. Refer to *4.20.3 Implementation Aspects* and *4.20.4 Configuration Aspects* for further details.

The DCM provides a means to specifiy the maximum length of the periodic frame for each diagnostic client. Refer to *4.20.3 Implementation Aspects* and *4.20.4 Configuration Aspects* for details about this setting.

FAQ
Only periodic responses of type 2 (UUDT) are supported, as the latest versions of [10] require.

4.20.2 Required Interfaces

- If service handled by DCM:
  - > *DataServices_<DataName>*
  - > *DataServices_DIDRange_<RangeName>*
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.20.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-39 Service 0x2A: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-40 Service 0x2A: Supported subservices

The protocol handling and the PDID read job scheduling of this service is fully implemented by DCM. The data reported by each DID will be provided by the application via service calls or call outs.

To reduce bus load, a delay can be induced between consecutive periodic messages. More specifically, a time after sending a periodic message can be configured in which at most one other periodic message may be sent. This effectively limits the number of periodic messages sent within this configured time to two messages. The time can be configured for each scheduling rate individually. Refer to *4.20.4 Configuration Aspects* for details on how this can be configured. In case multiple periodic requests with different scheduling rates are active simultaneously, the fastest one determines the delay time.

At the start of Service 0x2A the UUDT size of the requested DIDs are calculated. The calculated size is checked against the maximum possible UUDT frame size for the specific connection. If the maximum possible UUDT frame size is exceeded, a negative response with NRC 0x14 (response to long) for the Service 0x2A request is sent back to the tester. To adapt the UUDT frame size individually for the different physical interfaces (e.g. CAN, CAN FD), the UUDT frame size can be configured for each connection. Refer to *4.20.4 Configuration Aspects* for details on how this can be configured.

Caution
If you intend to use DID ranges, please read carefully chapter *8.20 Handling with DID Ranges* to learn important particularities.

4.20.4 Configuration Aspects

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container. The scheduling rates to be supported are specified by the corresponding rate time configuration parameter. Example for “SlowRate”: /Dcm/DcmConfigSet/DcmDsp/DcmDspPeriodicTransmission/DcmDspPeriodicTransmissionSlowRate

> The time window in which at most two messages are allowed to be sent can be configured for each scheduling rate individually. Example for “SlowRate”: /Dcm/DcmConfigSet/DcmDsp/DcmDspPeriodicTransmission/DcmDspPeriodicTransmissionDelaySlowRate

- All to be supported readable PDIDs shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspDid. The only allowed DID numbers are within the range [0xF200-0xF2FF].
- The read operation over a PDID is defined by: /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidRead
- The maximum number of simultaneously requested PDIDs shall be defined by: /Dcm/DcmConfigSet/DcmDsp/DcmDspMaxDidToRead
- The maximum number of simultaneously schedulable PDIDs shall be defined by: /Dcm/DcmConfigSet/DcmDsp/DcmDspPeriodicDidTransmission/DcmDspMaxPeriodicDidScheduler
- There shall be at least one DCM periodic connection (at least once client supports periodic responses), referred by a corresponding tester main connection. For that purpose, configure:
  - Define the client's periodic connection with one or multiple PDUs of the UUDT messages to be sent within the protocol container: /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslPeriodicTransmission
  - Refer the above created connection from the client's main connection located in the same protocol container: /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslPeriodicTranmissionConRef
- If it is required that DCM shall stop automatically any PDID which is no more supported in the active diagnostic session/security level the following parameter shall be enabled: /Dcm/DcmConfigSet/DcmDsp/DcmDspPeriodicDidTransmission/DcmDspPeriodicDidStopOnStateChange
- The maximum frame size for UUDT messages shall be defined for each individual diagnostic client by: /EcuC/EcucPduCollection/Pdu/PduLength
  If the configuration parameter is set to 0, all requests are accepted.

Caution
If a diagnostic client uses more than one periodic UUDT connection, each UUDT connection must be specified with the same PduLength.

4.21 DynamicallyDefineDataIdentifier (0x2C)

4.21.1 Functionality

This service is used to define new abstract data structures (DIDs) that refer to other statically configured DIDs or memory areas. The data structures are accessible for reading only through their assigned DDID (DynamicDID).

Optionally, DCM can clear automatically any already defined DDID that cannot be accessed any more, after a diagnostic session/security access level change. This also implies that a periodic DDID will be also removed from the periodic scheduler (*ReadDataByPeriodicIdentifier (0x2A)*). Refer to *4.21.4 Configuration Aspects* for details about this setting. Furthermore, DCM can store Dynamic Defined DID data/states in a non-volatile memory. Refer to *8.40 How to Persist Dynamic Defined DIDs* for details about this setting.

4.21.2 Required Interfaces

- If service handled by DCM:
  - > No additional interfaces are required for this service since it is completely handled within the DCM.
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.21.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID	■			

Table 4-41 Service 0x2C: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x01	■			
0x02	■			
0x03	■			
0x04 … 0xFF				■

Table 4-42 Service 0x2C: Supported subservices

This service is fully implemented by DCM corresponding to the [10].

Caution
If you intend to use DID ranges, please read carefully chapter *8.20 Handling with DID Ranges* to learn important particularities.

4.21.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

> If this service is to be used, there shall be at least one DID in the DCM configuration, determined as a DDID by the parameter:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidDynamicallyDefined

> If the objects (DIDs or memory areas) referenced by a DDID shall be validated against session, security and mode-rule preconditions each time the DDID is to be read:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidDynamicallyDefined

> DCM always verifies the session, security and mode-rule preconditions of a DDID when it is requested by a diagnostic client. You can configure DCM additionally to also check the objects (DIDs or memory areas) referenced by a DDID against their preconditions by parameter:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDDDidCheckPerSourceDid

> You can configure DCM to execute all in a DDID contained DID's *ConditionCheckRead()* operations when the DDID is requested by diagnostic client:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDDDidCheckConditionReadPerSourceDid

> If it is required DCM to automatically clear any no more supported in the active diagnostic session/security level DDID (and stop it from periodic reading), the parameter shall be enabled:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDDDidClearOnStateChange

FAQ
Enabling DcmDspDDDidClearOnStateChange does imply that any DDID access precondition evaluation for reading it once (*ReadDataByIdentifier (0x22)*) or periodically (*ReadDataByPeriodicIdentifier (0x2A)*) will not be performed. The reason is that once there is a change of the current diagnostic session/security access level, the DDID will no more exist and the ECU will reject any read request for it by NRC 0x31 (RequestOutOfRange). Combining this feature together with DcmDspDDDidCheckPerSourceDid increases the overall run time usage but also the access precondition dependent level safety.

4.22 WriteDataByIdentifier (0x2E)

4.22.1 Functionality

This service provides write access to predefined and marked by identifier data structures within the ECU.

4.22.2 Required Interfaces

- If service handled by DCM:
  - *DataServices_<DataName>*
  - *DataServices_DIDRange_<RangeName>*
  - *DataServices*
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

4.22.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-43 Service 0x2E: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-44 Service 0x2E: Supported subservices

The protocol handling of this service is fully implemented by DCM. The functionality for writing the data of each DID will be provided by the application via service calls or call outs.

Caution
If you intend to use DID ranges, please read carefully chapter *8.20 Handling with DID Ranges* to learn important particularities.

4.22.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> All to be supported writeable DIDs shall be defined within the following container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDid

> The write operation over a DID is defined by:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidWrite

> For each DID data signal the corresponding container shall be configured:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData

> For NvRam signal access select the value USE_BLOCK_ID in the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort

  > A NvRam block Id must be referenced:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataBlockIdRef

4.23 InputOutputControlByIdentifier (0x2F)

4.23.1 Functionality

This service provides IO control access to predefined and marked by identifier IO structures (ports) within the ECU.

4.23.2 Required Interfaces

- If service handled by DCM:
  > *DataServices_<DataName>*
  > *DataServices*
  > *IOControlRequest*
  > *IOControlResponse*
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.23.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-45 Service 0x2F: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-46 Service 0x2F: Supported subservices

The protocol handling of this service is fully implemented by DCM. The control functionality over the corresponding IO port will be performed by the application via service calls or call outs.

DCM monitors all IO DIDs put under control, once a requested IO control operation other than *ReturnControlToECU()* was successfully executed. This allows DCM to automatically reset the IO DID operations, calling their the *ReturnControlToECU()* operations once one of the following events occurs:

> A state transition to the Default diagnostic session

> A state transition to any diagnostic session, where the monitored IO DID is not supported

> A state transition to any security level, where the monitored IO DID is not supported

FAQ
If an IO DID is configured not to support operation *ReturnControlToECU()*, the automatic resetting of this IO DID is not supported. The application shall catch the mode switch for *DcmDiagnosticSessionControl* and reset this IO DID by itself.

Caution
Although it is allowed to have an asynchronous IO DID "*DataServices_<DataName>*" service port, it is not allowed to implement the "*ReturnControlToECU()*" operation of this port as asynchronous. This is because the transition to the default session is a synchronous operation and cannot be delayed.

If the DET support in DCM is enabled and you have implemented the "*ReturnControlToECU()*" operation to return DCM_E_PENDING, then this will cause a DET report.

4.23.3.1 IO DID Data Handling in DCM and Application

According to [10] there are two types of IO DIDs: packeted and bitmapped. The difference is the size of the IO signals addressed by an IO DID:

> Packeted: Each data element within the IO DID can be of any size.

> Bitmapped: Each data element within the IO DID has a size of a single bit.

For C/S DID data access, DCM can address only at least a whole byte element. So, there are two scenarios in using IO DIDs in DCM also regarding the CEMR:

> Packeted IO DID with all signals which have a size of a multiple of eight bits

> Bitmapped IO DID or IO DID where the signal size is not a multiple of eight bits

These two scenarios are described in detail in the next paragraphs.

4.23.3.2 Packeted IO DID with signals having a size of a multiple of eight bits

If the IO DID has multiple data signals, the DCM can automatically derive an appropriate CEMR for this DID as specified in [10]. Then at runtime during processing a valid request of this service, the DCM will call only the service ports of the IO DID that are enabled in the requested CEMR. To learn about how the automatic CEMR derivation can be enabled resp. disabled, please refer to *4.23.4 Configuration Aspects* and the detailed parameter description in the configuration tool.

FAQ
The CEM has only effect on the requested IO control operation. The returned data in the positive response will contain all IO DID data independently of the CEM value.

4.23.3.3 Bitmapped IO DID or IO DID where the signal size is not a multiple of eight bits

If the IO DID shall contain only single bit information or in general any data element of size not filling a complete byte, word etc., then such an IO DID must be represented by a single data object, which combines all the IO signals, including any reserved gaps in between or at the end of the DID.

If this DID shall support in addition also the CEMR, then it shall be specified to support the CEMR as a one handled by the application. To learn about how to specify an externally handled CEMR, please refer to *4.23.4 Configuration Aspects* and the detailed parameter description in the configuration tool.

4.23.4 Configuration Aspects

> This service shall be defined in the configuration tool:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined within the above defined service container

> All to be supported writeable DIDs shall be defined within the following container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDid

> Which IO operation is supported by the IO DID is defined within the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl
There you can create the operation corresponding sub-containers like:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl
/DcmDspDidShortTermAdjustment

> For each DID data signal the corresponding container shall be configured:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData

> Whether the IO DID shall support CEMR and which kind of CEMR (internal/external) handling is required, can be specified using parameter:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl
/DcmDspDidControlMask

> If a CEMR handled by the application shall be supported, its size shall be specified by the parameter:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl
/DcmDspDidControlMaskSize

> **Caution**
> If the IO DID has read operation (/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidRead, i.e. accessible via *ReadDataByIdentifier (0x22)*) the positive response to this service will return the actual IO data immediately after the request IO control operation was successfully applied. **Otherwise no response data will be returned.**

FAQ

Particularities of an IO DID configuration:

- An IO DID with read operation shall never have "*ConditionCheckRead()"* operation. For details, please refer to *4.14.4 Configuration Aspects* of service *ReadDataByIdentifier (0x22)*. The reason for that requirement is that the read operation is executed always after the IO control operation is applied. Once it is applied, the read operation must succeed and return the actual data. Otherwise the IO control operation must be undone and the response will be a negative one, which contradicts the IO control definition.
- If an IO DID has more than one data signal, DCM will automatically enable the "enable mask record" support for this DID (AR 4.0.3 requirement). But if you have configured only one signal for an IO DID and that signal actually represents all IO signals that the concrete IO DID indeed represents (e.g. for optimization purposes combined into a single byte stream), then you must use the /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl/DcmDspDidControlMask and /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidIoEnableMaskSize parameter in order to configured appropriate enable mask records size.
- An externally handled CEMR is passed to the application (refer to the corresponding operations of *DataServices_<DataName>* C/S interface*)* in exactly the same form as it was located in the request message: Always aligned with the MSB of the function argument:
  - For 8, 16 and 32bit CEMRs, the corresponding uint8/16/32 data type will be used as `<ControlMaskType>` to transfer the value to the application. It represents directly the CEMR from the request, starting with the MSB for the very first data element in the IO DID.
  - For a 24bit CEMR, DCM transfers the CEMR to the application using the uint32 data type for the `<ControlMaskType>`. In this case, to keep the bit scanning algorithm in the application consistent (i.e. shift left and extract bit) once again the MSB (and not bit 23 of the function argument value represents the very first data element).
  - For CEMRs with more than 32bits, the ControlMask function argument points to the first byte (MSB) of the requested CEMR using a uint8 data pointer (uint8*) as `<ControlMaskType>`.

4.24 RoutineControl (0x31)

4.24.1 Functionality

This service provides direct access to routines within the ECUs (e.g. self-test, control of peripheries, etc.).

4.24.2 Required Interfaces

- If service handled by DCM:
  - > *RoutineServices_<RoutineName>*
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*

4.24.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-47 Service 0x31: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-48 Service 0x31: Supported subservices

The protocol handling of this service is fully implemented by DCM, except the sub-function execution sequence validation (e.g. prior executing “stop” or “request results” there shall be send a “start” command).

Those sequence rules may not apply to all routines. Instead the application can implement an own state machine to model the running state of each routine. If the service execution order is not correct, the appropriate NRC (i.e. 0x24) can be returned from the corresponding service port, implemented by the application.

DCM does not support the automatic stop of all active routines on session transition. The application can catch the mode switch for DcmDiagnosticSessionControl and stop the routines by itself.

4.24.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined under the service container

> All to be supported RIDs shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine

> The sub-function to be supported by a RID is to be specified within the concrete RID container (sub-function “start” is always available): /Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine

> Overwrite protection of input and output data (Interface Argument Integrity): /Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDsp<RoutineControlType>InterfaceArgumentIntegrity

4.24.5 Request and Response Data Pointers in an Operation for RIDs

According to chapter 3.4 in a callout the application might overwrite its input data when writing output data. In the diagnostic service RoutineControl (0x31) it can be configured whether the DCM shall provide an additional internal buffer to separate request and response data.

If Interface Argument Integrity is active the response data cannot overwrite the request data, that is there is no need to store the request data temporary in the application software. This applies to RTE port operation functions as well as simple C callouts.

FAQ

Particularities for OBD RIDs (i.e. all within [0xE000-0xE1FF]):

> Any OBD availability RID (e.g. 0xE000, 0xE020, 0xE100, 0xE1A0, etc.) will always be implemented by DCM. They will return the corresponding RID availability mask value as described in [12].

> Every RID in the OBD range that covers a corresponding OBD TID, shall not contain any data definition. The concrete data will be processed out by DCM directly using the corresponding OBD TID service data access method. For such RIDs, there also will be no RTE RoutineServices port or callback generated.

> Only “StartRoutine” operation is to be used on OBD RIDs, since [12] does not define any other operation over a RID.

> Any OBD RID, that neither is an availability RID, nor covers any existing OBD TID, will be handled as a generic RID and shall be configured regularly.

4.25 RequestDownload (0x34)

4.25.1 Functionality

This service provides the functionality to initiate a download data transfer from the client to the server.

4.25.2 Required Interfaces

The following interface must be available when service 0x34 is internally used:

> *ProcessRequestDownload()*

▶ If service handled by the application:

> *<Module>_<DiagnosticService>()*

4.25.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-49 Service 0x34: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-50 Service 0x34: Supported subservices

This service is fully implemented by DCM. This includes:

> Validating and evaluating the memory identifier byte

> Validating and evaluating the ALFID byte

The request for a download will be then provided by the application by a call out.

4.25.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined under the service container

> All to be supported memory identifier and ALFIDs shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspMemoryTransfer

4.26 RequestUpload (0x35)

4.26.1 Functionality

This service provides the functionality to initiate an upload data transfer from the server to the client.

4.26.2 Required Interfaces

The following interface must be available when service 0x35 is internally used:

> *ProcessRequestUpload()*

If service handled by the application:

> *<Module>_<DiagnosticService>()*

4.26.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-51 Service 0x35: Implementation types

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
all				■

Table 4-52 Service 0x35: Supported subservices

This service is fully implemented by the DCM. This includes:

> Validating and evaluating the memory identifier byte
> Validating and evaluating the ALFID byte

The request for an upload will be then provided to the application by a call out.

4.26.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined under the service container

All to be supported memory identifier and ALFIDs shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspMemoryTransfer

4.27 TransferData (0x36)

4.27.1 Functionality

This service provides the functionality to transfer data from the client to the server (download) and from the server to the client (upload).

4.27.2 Required Interfaces

The following interface must be available when service 0x36 is internally used:

> *ProcessTransferDataWrite()*

> *ProcessTransferDataRead()*

▶ If service handled by the application:

> *<Module>_<DiagnosticService>()*

4.27.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-53 Service 0x36: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-54 Service 0x36: Supported subservices

This service is partially implemented by DCM. Currently there is the functionality to download data from the client to the server and upload data from the server to the client available.

4.27.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined under the service container

4.28 RequestTransferExit (0x37)

4.28.1 Functionality

This service provides the functionality terminate a data transfer between the client and the server.

4.28.2 Required Interfaces

The following interface must be available when service 0x37 is internally used:

> *ProcessRequestTransferExit()*

▶ If service handled by the application:

> *<Module>_<DiagnosticService>()*

4.28.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-55 Service 0x37: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-56 Service 0x37: Supported subservices

This service is fully implemented by DCM.

4.28.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> No sub-functions shall be defined under the service container

4.29 WriteMemoryByAddress (0x3D)

4.29.1 Functionality

This service provides direct write access to the physical memory of the ECU. All writeable memory areas and their access preconditions are to be configured as documented in *4.15.4 Configuration Aspects*.

4.29.2 Required Interfaces

- If service handled by DCM:
  - *Dcm_WriteMemory()*
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

4.29.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID				■

Table 4-57 Service 0x3D: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
all				■

Table 4-58 Service 0x3D: Supported subservices

The protocol handling of this service is fully implemented by DCM. This includes:

- Validating and evaluating the ALFID byte
- Parsing the requested memory address and size parameters
- Validating the requested memory block against the DCM memory configuration:
  - Supported requested memory area by the ECU
  - Memory area access preconditions (e.g. security access, mode rules)

The memory access will then be provided by the application via a call out.

4.29.4 Configuration Aspects

- This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService
- No sub-functions shall be defined within the above defined service container.
- All to be supported writeable memory ranges shall be defined within the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspMemory

  They can be configured either using numerical or symbolic representations.

> For numerical representation:
/Dcm/DcmConfigSet/DcmDsp/DcmDspMemory/DcmDspMemoryIdInfos/DcmDspMemoryIdInfo/DcmDspWriteMemoryRangeInfos

> For symbolic representation:
/Dcm/DcmConfigSet/DcmDsp/DcmDspMemory/DcmDspMemoryIdInfos/DcmDspMemoryIdInfo/DcmDspWriteMemoryRangeByLabelInfos

The header file, containing the definition of the labels, can be added by
/Dcm/DcmConfigSet/DcmGeneral/DcmHeaderFileInclusion

Caution
Memory ranges represented by symbols (DcmDspWriteMemoryRangeByLabelLow resp. DcmDspWriteMemoryRangeByLabelHigh) shall not overlap each other and have correct relationships between start and end addresses (start address <= end address).

Note
It can be decided individually per memory identifier (DcmDspMemoryIdInfo) whether the memory ranges are represented numerically or symbolically.

4.30 TesterPresent (0x3E)

4.30.1 Functionality

This service is only used for keeping the current diagnostic state in the ECU active. Otherwise on lack of diagnostic communication, the ECU will reset all temporary activated states and functionalities (e.g. diagnostic session, security access, routine execution, etc.)

4.30.2 Required Interfaces

- If service handled by DCM:

> No interfaces required for this service.

- If service handled by the application:

> *<Module>_<DiagnosticService>()*

4.30.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID	■			

Table 4-59 Service 0x3E: Implementation types

Subservice ID \ Implementation	internal only	internal or external	external only	not allowed
0x00	■			
0x01 … 0xFF				■

Table 4-60 Service 0x3E: Supported subservices

This service is fully implemented by DCM but can be also handled by the application.

Caution
If you intend to handle this service within your application, please be aware that the application callback will be called for any request for this service except the "**functionally requested 0x3E 0x80**"! The latter is always handled within DCM.

4.30.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

Caution
This service is mandatory and therefore may not be missing in the configuration.

4.31 ControlDTCSetting (0x85)

4.31.1 Functionality

This service manipulates the setting of the DTC in the ECU to avoid unnecessary fault memory entries (i.e. while the communication is disabled).

4.31.2 Required Interfaces

- If service handled by DCM:
  > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  > *<Module>_<DiagnosticService>()*

4.31.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID	■			

Table 4-61 Service 0x85: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00				■
0x01 … 0x02	■			
0x03 … 0xFF				■

Table 4-62 Service 0x85: Supported subservices

This service is completely implemented by DCM.

4.31.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

> If DCM shall also accept a DTC group as a request parameter for this service, please enable the following option: /Dcm/DcmConfigSet/DcmDsp/DcmDspControlDTCSetting/DcmSupportDTCSettingControlOptionRecord

> In case DCM shall monitor any critical condition under which this service shall no longer be active, put a reference to that condition using parameter: /Dcm/DcmConfigSet/DcmDsp/DcmDspControlDTCSetting/DcmDspControlDTCSettingReEnableModeRuleRef

4.32 ResponseOnEvent (0x86)

4.32.1 Functionality

This service is implemented according to the revised version of [11] and used for monitoring of certain ECU internal diagnostic events such as DTC status changed and triggering a

reporting action in form of a diagnostic service to the client that has requested the event monitoring.

4.32.2 Required Interfaces

The following interfaces must be available when service 0x86 is used:

- If service handled by DCM:
  - > Refer to chapter *5.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - > *<Module>_<DiagnosticService>()*
  - > *Dcm_ProcessVirtualRequest()*

4.32.3 Implementation Aspects

Implementation / Protocol Level	internal only	internal or external	external only	not allowed
ServiceID		■		
SubServiceID	■			

Table 4-63 Service 0x86: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00, 0x01, 0x03, 0x05, 0x06, 0x07, 0x09	■			
All other				■

Table 4-64 Service 0x86: Supported subservices

The internally handled sub-functions completely cover the service handling with following limitations and specifics:

- > Currently only EWT 0x02 (Infinite) is supported

Note
The parameter EWT is only evaluated at request for sub-function 0x05.

- > The event types supported are:
  - > “*OnDtcStatusChange*” (sub-function 0x01)
  - > “*OnChangeOfDataIdentifier*” (sub-function 0x03)

> “*OnComparisonOfValues*” (sub-function 0x07)

> “*ReportDTCRecordInformationOnDtcStatusChange*” (sub-function 0x09)

Please refer to sub section 4.32.3.1, 4.32.3.2, 4.32.3.3, and 4.32.3.4 for more details.

FAQ
If the STRT service shall only be available for RoE but not accessible for external testers, then the following restrictions shall be considered:

> The STRT service must exist in the DCM configuration within the service table, where the RoE service is available. Otherwise, client prioritization and STRT accessibility will not work (i.e. the client that starts the RoE and wants to get this STRT shall belong to the ProtocolRow that refers to the service table containing the STRT).

> Depending on the service restriction level (i.e. SID or sub-service ID), a SW-C or CDD callout for *ServiceRequestManufacturerNotification_<SWC>* resp. *ServiceRequestSupplierNotification_<SWC>* shall be configured for DCM that will decide whether the STRT shall pass to the DCM dispatcher or shall be rejected with an appropriate NRC. For that purpose the implementation shall be able to distinguish between “external” and “internal (RoE)” requests and that shall be done using the provided DCM service *Dcm_GetRequestKind()*.

Note
If a STRT related to a low priority tester is triggered while a service request from a high priority tester is already in process, then DCM will either ignore the STRT or reject it by NRC 0x21 (busyRepeatRequest). In this scenario it might be possible that an unsolicited NRC 0x21 is sent to the low priority tester. The concrete reaction depends on the setting:

/Dcm/DcmConfigSet/DcmDsl/DcmDslDiagResp/DcmDslDiagRespOnSecondDeclinedRequest

Note
Manufacturer indication is needed to ensure that only one tester can use RoE at a time assuming that it is not already ensured by proper service table and protocol configuration.

4.32.3.1 OnDtcStatusChange

- DTC event detection is triggered on any transition of the related DTC status bits, i.e. 0→1 and 1→0 for any bit specified in the request DTC status mask. The STRT is restricted to service 0x19. Since this is a service with sub-functions, the minimum length for the STRT is therefore two.
- In order to get the RoE monitoring of DTC status change working, the DEM shall implement the notification service “*Dcm_DemTriggerOnDTCStatus()*” as per [1], i.e.:

  > Notify DCM on DTC status change only when requested (DCM has enabled the monitoring in DEM by calling

"**Dem_DcmControlDTCStatusChangedNotification()**" with parameter value **TRUE**

> Do not notify DCM for changed DTC status while executing "**Dem_XxxClearDTC()**"

4.32.3.2 OnChangeOfDataIdentifier

DID event detection is triggered on any new data record identified by data identifier. If multiple onChangeOfDataIdentifier events are running simultaneously, the ECU temporarily stops all event logic while the event reporting is handled for a given event. A First-In First-Out (FIFO) queue is not used. The STRT is restricted to service 0x22 with at least one DID, which may be different from that one in the eventTypeRecord.

Note
Since the ECU temporarily stops all event logic while the event reporting is handled, it is not recommended to setup multiple onChangeOfDataIdentifier events for the same DID in the eventTypeRecord. Otherwise, DCM would send only a single STRT for the event configured first. Instead of that, a single onChangeOfDataIdentifier event should be setup with a STRT containing multiple DIDs.

4.32.3.3 OnComparisonOfValues

Comparison of values event is triggered on the change of specific data record of data identifier in comparison to a predefined comparison value. This comparison of the data record is done with the help of the following four main parameters:

- Comparison logic such as comparison parameter value is bigger than measured value.
- Raw reference comparison value which is the value to be compared to.
- Hysteresis value which specifies a certain percentage of the comparison value as a limit to trigger the event.
- Localization parameter which contains the sign of the comparison, the length data record of data identifier to be compared, and the offset from where to extract data record from the data identifier.

If multiple OnComparisonOfValues events are running simultaneously, the ECU temporarily stops all event logic while the event reporting is handled for a given event. A First-In First-Out (FIFO) queue is not used. The STRT is restricted to service 0x22 with at least one DID.

Note
Since the ECU temporarily stops all event logic while the event reporting is handled, it is not recommended to setup multiple OnComparisonOfValues events for the same DID in the eventTypeRecord. Otherwise, DCM would send only a single STRT for the event configured first. Instead of that, a single OnComparisonOfValues event should be setup with a STRT containing multiple DIDs.

Note
DCM is not in charge to detect invalid hysteresis values by responding with NRC 0x31 e.g., if hysteresis is 50% for a 8-bit value of 200 that will cause overflow to the upper limit value as the 300 cannot fit the 8-bit size

> For signed comparison, DCM accepts only the basic data type bit sizes 8, 16 and 32.

4.32.3.4 ReportDTCRecordInformationOnDtcStatusChange

The triggering mechanism of this event type is identical to that of *OnDtcStatusChange*. As a result, it is required to implement the same prerequisite in *OnDtcStatusChange* when this event type is to be enabled in the configuration. The STRT is fixed to service 0x19 and it is not necessary to define it in the request message. The supported sub-functions of the STRT are 0x04, 0x06, 0x09, 0x10, 0x18, and 0x19.

Note
As a full STRT request message is not expected in the request message for this sub-function as per [11], the positive response message for the request does not contain the full STRT request definition either.

In contrast to *OnDtcStatusChange*, Dcm is able to identify multiple events according to the configured DTC status mask. These events are stored in an internal queue such that a series of STRTs for these events can be triggered afterwards. When more events than the MaxNumberOfStoredDTCStatusChangedEvents as defined in [11] have been identified, Dcm will not be able to store all identified events. Under this circumstance, in addition to the STRT responses for the stored events, an intermediate response is triggered to signal the tester to synchronize the status of the monitored DTCs.

4.32.3.5 Pause RoE Processing

In some cases it might be desirable to pause the processing of RoE events. E.g. the ECU is powered in ignition-off state but diagnostics would still be possible. However, to safe the vehicle battery, the network shall not be woken up by RoE messages.

For these cases, the application can pause the processing of RoE events and sending of STRT by calling *Dcm_SetRoeActivationState()* with the according state.

Note
The activation state is not persisted. At startup the RoE activation state is always active.

Note
Only STRT and the trigger are suppressed. It is still possible to start/stop/clear/setup RoE with the according subfunction.

Note
If any event processing was started before the state is deactivated, that event processing is finished. Nevertheless, the STRT is suppressed.

Note
A queue for subfunction 0x09 is emptied passively. So, if the state is disabled and after a short period of time reactivated, it might happen, that the STRT for the first events in the queue were suppressed but the last events trigger the STRT on the bus.

4.32.4 Configuration Aspects

The following configuration parameter shall be considered for the proper DCM function on this service.

> This service shall be defined in the configuration tool: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

> All to be supported sub-functions shall be defined within the above defined service container as sub-service containers: /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService

> All to be supported EWT values shall be defined within the DCM ECUC container: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeEventWindowTime

> If MICROSAR Classic DEM is used together with this DCM, on order to get this service working for onDTCStatusChange event type, the DEM shall be configured to notify DCM for a DTC status change via the API *Dcm_DemTriggerOnDTCStatus()*. Please refer to [19] or DEM ECUC configuration help text to get information on how to perform this configuration step

> The ResponseOnEventSchedulerRate setting according to [11] shall be defined here: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeSchedulerRate

> If applicable, the MaxNumChangeOfDataIdentfierEvents setting according to [11] shall be defined here: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeMaxNumChangeOfDidEvents

> If applicable, the MaxSupportedDIDLength setting according to [11] shall be defined here: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeMaxSupportedDidLength

- The maximum number of allowed DIDs in the STRT is derived from the following configuration option: /Dcm/DcmConfigSet/DcmDsp/DcmDspMaxDidToRead
- The maximum number of events that can be simultaneously configured with sub-function onComparisonOfValues (0x07) shall be defined here: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeMaxNumComparisonOfValueEvents
- The maximum number of identified events that can be stored for setting up the sub-function ReportDTCRecordInformationOnDtcStatusChange (0x09) shall be defined here: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeMaxNumStoredDTCStatusChangedEvents
- To support the storageState “storeEvent” (sub-function 0x45), the following configurations are required:

  The following parameter has to be enabled: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeStorageStateStoreEventEnabled

  A RoE NvM block reference by parameter has to be defined: /Dcm/DcmConfigSet/DcmDsp/DcmDspRoe/DcmDspRoeBlockIdRef

  Furthermore, the parameters NvMBlockUseSetRamBlockStatus and NvMSelectBlockForWriteAll shall be set to TRUE: /NvM/NvMBlockDescriptor/NvMBlockUseSetRamBlockStatus, /NvM/NvMBlockDescriptor/NvMSelectBlockForWriteAll

  The RAM block data must be set to the symbol “Dcm_Svc86NvMData“: /NvM/NvMBlockDescriptor/NvMRamBlockDataAddress

  The ROM block data must be set to the symbol “Dcm_Svc86DefaultNvMData“: /NvM/NvMBlockDescriptor/NvMRomBlockDataAddress

  For the NvM RoE storage the DCM defines the following data type: `Dcm_ExtSvc86NvMDataType`

  Use this data type for the NvM block size calculation.
- All other options of DcmDspRoe have no effect

Note
Sub-function onChangeOfDataIdentifier requires the server to sample and compare the value of the configured DID. The DID data reference value for comparison is gathered during the next call of the RoE scheduler. The scheduler itself is started immediately after processing of sub-function startResponseOnEvent. If an event is added while RoE is already started, it will take up to one RoE scheduling cycle to be able to recognize changes of DID values according to [11]. This in turn means, that the first STRT can only be sent after one RoE scheduling cycle at the earliest.

5 API Description

For an interface overview please see *Figure 1-2*.

5.1 Type Definitions

All types not described here are defined by the DCM as described in [1].

5.1.1 Dcm_ProtocolType

Type Name	C-Type	Description	Value Range
Dcm_ProtocolType	uint8	Specifies the currently active protocol in DCM.	`[0x00-0x0B]∪[0xF0-0xFE]` These values are defined in [1].
			`DCM_NO_ACTIVE_PROTOCOL (0x0C)` No protocol has been activated yet.

Table 5-1 Dcm_ProtocolType

5.1.2 Dcm_RecoveryInfoType

Struct Element Name	C-Type	Description	Value Range
CommControlState	uint8 [M] (typically)	List of all DCM ComControl related (internal handle, no ComM SNV representation) channels with value equal to the corresponding enumeration type. Exist-Condition: *CommunicationControl (0x28)* is supported in DCM.	`DCM_ENABLE_RX_TX_NORM_NM` - DCM will not perform any CommunicationControl operation.
			`Any other` - DCM will perform the corresponding CommunicationControl operation on the corresponding channel.
ComMChannelState	Boolean [N]	List of all DCM related (internal handle, no ComM SNV representation) channels. If a non-default session shall be started, this list must exist to start up all affected ComM channels.	`[X] = FALSE` – DCM will leave the ComM channel in its default state.
			`[X] = TRUE` – DCM will activate the ComM on that channel.
ControlDTCSettingDTCGroup	uint32	Optional parameter in case service *ControlDTCSetting (0x85)* is enabled in DCM and supports DTC group parameter.	`<DTCgroup>` - The DTC group that shall be used for the ControlDTCSetting API in DEM.
ControlDTCSettingDisabled	boolean	The new ControlDTCSetting state. Exist-Condition: *ControlDTCSetting (0x85)* is enabled in DCM	`FALSE` – DCM will not call the ControlDTCSetting DEM API.
			`TRUE` - DCM will perform a ControlDTCSetting operation for “disabling DTC” as for an

Struct Element Name	C-Type	Description	Value Range
			external diagnostic request for *ControlDTCSetting (0x85)*.
SessionLevel	uint8 (typically)	New diagnostic session. Note: This is not the session level as defined by AR. It is a DCM internal value.	`0` – DCM will stay in the default session.
			`Any other valid value` DCM will perform a session transition as if the corresponding request has been received.
SecurityLevel	uint8 (typically)	New security level. Note: This is not the security level as defined by AR. It is a DCM internal value. Exist-Condition: If *SecurityAccess (0x27)* is supported in DCM.	`0` – DCM will stay in the locked state.
			`Any other valid value` DCM will perform a security level transition as if the corresponding request has been received.
SessionConnection	uint8 (typically)	Transfers the client connection ID (internal DCM value) that has started the non-default session. Exist-Condition: Only if non-default session protection against other clients is required.	`Any value` – Proper connection ID on the last client started the non-default session.
ActiveProtocol	uint8	New active protocol. Note: This is not the protocol ID as defined by AR. It is a DCM internal value.	`Any value` – Proper protocol ID of the last started protocol.
Signature	uint32	Magic number for data consistency check between stored and to be recovered data block.	A configuration dependent value.
SessionClientSrcAddr	uint16	Source address of client which initiated the non-default session	Generic connection: 0x00 to 0xFF. Non-generic connection: 0x0000 to 0xFFFF.

Table 5-2 Dcm_RecoveryInfoType

5.1.3 Dcm_VsgIdentifierType

Type Name	C-Type	Description	Value Range
Dcm_VsgIdentifierType	uint8/ uint16	Unique Identifier of a Vehicle System Group (VSG). Note: C-Type depends on the total number of VSGs. If number of VSGs is more than 255 the C-Type is uint16, otherwise it is uint8.	[1-65535]

Table 5-3 Dcm_VsgIdentifierType

Caution
Changing the configured number of VSGs can change the C-Type of Dcm_VsgIdentifierType. Already mapped SWC ports will be disconnected if the C-Type changes (see *5.6.1.1*).

5.1.4 Dcm_VsgStateType

Type Name	C-Type	Description	Value Range
Dcm_VsgStateType	uint8	Allowed states of a Vehicle System Group (VSG).	DCM_VSG_ENABLED VSG is enabled or shall be enabled. DCM_VSG_DISABLED VSG is disabled or shall be disabled.

Table 5-4 Dcm_VsgStateType

5.1.5 Dcm_InputOutputControlParameterType

C-Type	Description	Value Range
uint8	First byte of controlOptionRecord in the request of service 0x2F.	DCM_RETURN_CONTROL_TO_ECU Indicates to the server that the client does no longer have control on signals.
		DCM_RESET_TO_DEFAULT Inidcates to the server that it is requested to reset signals.
		DCM_FREEZE_CURRENTLY_STATE Indicates to the server that it is requested to freeze current state of signals.
		DCM_SHORT_TERM_ADJUSTMENT Indicates to the server that it is requested to adjust signals.
		DCM_IDLE

		Indicates that server is Idle, no request in processing (initial value).

Table 5-5 Dcm_inputOutputControlParameterType

5.1.6 Dcm_IOOperationResponseType

C-Type	Description	Value Range
uint8	Positive and negative response codes for sevice 0x2F.	DCM_POSITIVE_RESPONSE Inidcates postive response.
		DCM_GENERAL_REJECT Indicates NRC general reject.
		DCM_BUSY_REPEAT_REQUEST Indicates NRC busy repeat request.
		DCM_CONDITIONS_NOT_CORRECT Indicates NRC conditions not correct.
		DCM_FAILURE_PREVENTS_EXECUTION Indicates NRC failure prevents execution of requested action.
		DCM_REQUEST_OUT_OF_RANGE Indicates NRC request out of range.
		DCM_RESPONSE_PENDING Indicates response pending (like DCM_E_PENDING).

Table 5-6 Dcm_IOOperationResponseType

5.1.7 Dcm_IOOperationRequest_< DidName/DidDataName >Type

Type Name	C-Type	Description	Value Range
Dcm_InputOutputControlParameterType	Refer to Table *5-5*		
Dcm_ControlMask_<xx>Type	uint8/ uint16/ uint32	C-Type depends on <xx> which has a value of either 8, 16 or 32.	Configuration dependent.

Table 5-7 Dcm_IOOperationRequest_< DidName/DidDataName >type

5.1.8 Dcm_SpecificCauseCodeType

Type Name	C-Type	Description	Value Range
Dcm_SpecificCauseCodeType	uint8	Specific cause code in case of negative response.	[0-255]

Table 5-8 Dcm_SpecificCauseCodeType

5.1.9 Dcm_AuthenticationInfoType

Type Name	C-Type	Description	Value Range
AuthRole	uint8/ uint16/ Uint32	New role which shall be set for the authenticated connection. Note: C-Type depends on DcmDspAuthenticationRoleSize.	Configuration dependent.
WhiteLists	Dcm_WhiteListContextType	Struct which holds the white list information. Exist-Condition: Member only exists, if any white list is configured.	

Table 5-9 Dcm_AuthenticationInfoType

5.1.10 Dcm_WhiteListContextType

This type only exists, if any white list is configured.

Type Name	C-Type	Description
WhiteListServiceContext	Dcm_WhiteListServiceContextType	Struct which holds the information for the service white list. Exist-condition: Member only exists, if a service white list is configured: DcmDspAuthenticationWhiteListServicesMaxSize > 0
WhiteListDidContext	Dcm_WhiteListDidContextType	Struct which holds the information for the Did white list. Exist-condition: Member only exists, if a did white list is configured: DcmDspAuthenticationWhiteListDIDMaxSize> 0

WhiteListRidContext	Dcm_WhiteListRidContextType	Struct which holds the information for the Rid white list. Exist-condition: Member only exists, if a rid white list is configured: DcmDspAuthenticationWhiteListRIDMaxSize> 0
WhiteListMemContext	Dcm_WhiteListMemContextType	Struct which holds the information for the Mem white list. Exist-condition: Member only exists, if a memory white list is configured: DcmDspAuthenticationWhiteListMemorySelectionMaxSize> 0

Table 5-10 Dcm_WhiteListContextType

5.1.11 Dcm_WhiteListServiceContextType

This type only exists, if a service white list is configured:

DcmDspAuthenticationWhiteListServicesMaxSize > 0

Type Name	C-Type	Description
NumServiceWhiteListElements	uint16	Number of actual service white list elements elements for the specific white list. Value Range: [0-DcmDspAuthenticationWhiteListServicesMaxSize]
WhiteListServiceElement	Dcm_WhiteListServiceElementContextType[M]	List of all service white list elements. Note: Size of list matches the maximum number of elements: DcmDspAuthenticationWhiteListServicesMaxSize

Table 5-11 Dcm_WhiteListServiceContextType

5.1.12 Dcm_WhiteListServiceElementContextType

This type only exists, if a service white list is configured:

DcmDspAuthenticationWhiteListServicesMaxSize > 0

Type Name	C-Type	Description
WhiteListData	uint8[N]	List with data for the actual service white list element. Note: Maximum size of list is 4. But the size for the actual element is defined by the following member.
WhiteListEntryLength	uint8	Length of the WhiteListData list. Maximum value is 4.

Table 5-12 Dcm_WhiteListServiceElementContextType

5.1.13 Dcm_WhiteListDidContextType

This type only exists, if a did white list is configured:

DcmDspAuthenticationWhiteListDIDMaxSize> 0

Type Name	C-Type	Description
NumDidWhiteListElements	uint16	Number of actual Did white list elements for the specific white list. Value range: [0 – DcmDspAuthenticationWhiteListDIDMaxSize / 3]
WhiteListDidElement	Dcm_WhiteListDidElementContextType[M]	List of all Did white list elements. Note: Size of list matches the maximum number of actual Did: DcmDspAuthenticationWhiteListDIDMaxSize / 3

Table 5-13 Dcm_WhiteListDidContextType

5.1.14 Dcm_WhiteListDidElementContextType

This type only exists, if a did white list is configured:

DcmDspAuthenticationWhiteListDIDMaxSize> 0

Type Name	C-Type	Description	Value Range
DidNumber	uint16	Actual Did which shall be white listed.	[0 – 65535]
DidAccessMask	uint8	Did access mask for Did which is white listed.	[0 – 7] – Bitmask Bit 0: Read Access Bit 1: Write Access Bit 2: Control Access

Table 5-14 Dcm_WhiteListDidElementContextType

5.1.15 Dcm_WhiteListRidContextType

This type only exists, if a rid white list is configured:

DcmDspAuthenticationWhiteListRIDMaxSize> 0

Type Name	C-Type	Description
NumRidWhiteListElements	uint16	Number of actual Rid white list elements for the specific white list. Value range: [0 – DcmDspAuthenticationWhiteListRIDMaxSize / 3]
WhiteListRidElement	Dcm_WhiteListRidElementContextType[M]	List of all Rid white list elements. Note: Size of list matches the maximum number: DcmDspAuthenticationWhiteListRIDMaxSize / 3

Table 5-15 Dcm_WhiteListRidContextType

5.1.16 Dcm_WhiteListRidElementContextType

This type only exists, if a rid white list is configured:

DcmDspAuthenticationWhiteListRIDMaxSize> 0

Type Name	C-Type	Description	Value Range
RidNumber	uint16	Actual Rid which shall be white listed.	[0 – 65535]
RidAccessMask	uint8	Rid access mask for Rid which is white listed.	[0 – 7] – Bitmask Bit 0: Access to “startRoutine” Bit 1: Access to “stopRoutine” Bit 2: Access to “requestRoutineResult"

Table 5-16 Dcm_WhiteListRidElementContextType

5.1.17 Dcm_WhiteListMemContextType

This type only exists, if a memory white list is configured:

DcmDspAuthenticationWhiteListMemorySelectionMaxSize> 0

Type Name	C-Type	Description
NumMemWhiteListElements	uint16	Number of actual memory selection white list elements for the specific white list. Value range: [0 – DcmDspAuthenticationWhiteListMemorySelectionMaxSize]
WhiteListMemElement	Dcm_WhiteListMemElementContextType[M]	List of all memory selection white list elements. Note: Size of list matches the maximum number: DcmDspAuthenticationWhiteListMemorySelectionMaxSize

Table 5-17 Dcm_WhiteListMemContextType

5.1.18 Dcm_WhiteListMemElementContextType

This type only exists, if a memory white list is configured:

DcmDspAuthenticationWhiteListMemorySelectionMaxSize> 0

Type Name	C-Type	Description	Value Range
MemNumber	uint8	Actual memory selection which shall be white listed.	[0 – 255]

Table 5-18 Dcm_WhiteListMemElementContextType

5.2 Services provided by DCM

5.2.1 Administrative

5.2.1.1 Dcm_Init()

Prototype	
`void Dcm_Init ( const Dcm_ConfigType *ConfigPtr )`	
Parameter	
`ConfigPtr`	The parameter specifies the configuration root the DCM shall use for this power on cycle. In case of pre-compile configuration – this parameter shall be NULL_PTR. If any other address is used, it will have no effect. In case of post-build selectable: > More than one variant is configured – the pointer shall be the address of one of the generated variant structures in Dcm_Lcfg.c > Only one variant is available – DCM is technically put into pre-compile mode (see above) In case of post-build loadable always a valid pointer to the root DCM structure shall be passed. In case of post-build selectable loadable always a valid pointer to the variant root structure shall be passed.
Return code	
`void`	N/A
Functional Description	
Service for basic initialization of DCM module. In all cases where this API does expect a non-null pointer argument, a validation of the passed argument is performed. For details on that topic, please refere to *8.19.2.1 Error Detection and Handling*.	
Particularities and Limitations	
> ServiceID: 0x01 > This function is not reentrant. > This function is synchronous.	

Table 5-19 Dcm_Init()

5.2.1.2 Dcm_MainFunction()

Prototype	
`void Dcm_MainFunction ( void )`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A

Functional Description
This service is used for processing the tasks of the main loop.
Particularities and Limitations
> ServiceID: 0x25 > This function is not reentrant. > This function is synchronous.

Table 5-20 Dcm_MainFunction()

5.2.1.3 Dcm_MainFunctionTimer()

Prototype	
`void` **`Dcm_MainFunctionTimer`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
This service is used for time critical tasks (high priority task).	
Particularities and Limitations	
> ServiceID: 0x25 > This function is not reentrant. > This function is synchronous.	

Table 5-21 Dcm_MainFunctionTimer()

5.2.1.4 Dcm_MainFunctionWorker()

Prototype	
`void` **`Dcm_MainFunctionWorker`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
This service is used for diagnostic service processing (low priority task). Note: All application call outs the DCM executes are performed only from within this task.	
Particularities and Limitations	
> ServiceID: 0x25 > This function is not reentrant. > This function is synchronous.	

Table 5-22 Dcm_MainFunctionWorker()

5.2.1.5 Dcm_GetVersionInfo()

Prototype	
`void` **`Dcm_GetVersionInfo`** `( Std_VersionInfoType *versionInfo )`	
Parameter	
`versionInfo`	Pointer to where to store the version information of this module.
Return code	
`void`	N/A
Functional Description	
Returns the version information of the used DCM implementation.	
Particularities and Limitations	
> ServiceID: 0x24 > This function is reentrant. > This function is synchronous.	

Table 5-23 Dcm_GetVersionInfo()

5.2.1.6 Dcm_InitMemory()

Prototype	
`void` **`Dcm_InitMemory`** `( void )`	
Parameter	
–	-
Return code	
`void`	N/A
Functional Description	
Service to initialize module global variables at power up. This function initializes the variables in DCM_VAR_INIT_* sections (refer to *3.3 Compiler Abstraction and Memory Mapping*) and shall be used in case they are not initialized by the startup code.	
Particularities and Limitations	
> This function must be called prior to *Dcm_Init()*. > This function is not reentrant. > This function is synchronous.	

Table 5-24 Dcm_InitMemory()

5.2.1.7 Dcm_ProvideRecoveryStates()

Prototype	
`Std_ReturnType` **`Dcm_ProvideRecoveryStates`** `( Dcm_RecoveryInfoType *RecoveryInfo )`	
Parameter	
`RecoveryInfo`	Contains all the information that must be stored for later recovery.
Return code	
`Std_ReturnType`	E_OK: Recovery info could be retrieved and now can be stored. E_NOT_OK: Some error occurred during state retrieval. Provided data is invalid and shall not be stored.
Functional Description	
This API shall be called by the DCM application right before performing the reset operation. For details on the usage of this API, please refer chapter *8.27 How to Recover DCM State Context on ECU Reset/Power On*. Note: > Once this API is called, the states may change due to external events (e.g. session timeout). Therefore, always perform this call right before executing the reset or within the context of a diagnostic service processing (i.e. before the final response is sent). For details on the recovered information, please refer the data type definition: *Dcm_RecoveryInfoType*.	
Particularities and Limitations	
> ServiceID: 0xA3 > This function is not reentrant. > This function is synchronous.	

Table 5-25 Dcm_ProvideRecoveryStates()

5.2.2 SWC

5.2.2.1 Dcm_GetActiveProtocol()

Prototype	
`Std_ReturnType` **`Dcm_GetActiveProtocol`** `( Dcm_ProtocolType *ActiveProtocol )`	
Parameter	
`ActiveProtocol`	Currently active protocol type.
Return code	
`Std_ReturnType`	E_OK: This value is always returned.
Functional Description	
This function returns the active protocol Id. Please note, that depending on the configuration the active protocol Id might change at any time during the call of *Dcm_MainFunctionWorker()*. See chapter *2.4.1 Split Task Functions* for more details. For an example, when a service request is being processed, the active protocol Id may be different during the call to the *Indication()* and *Confirmation()* functions.	
Particularities and Limitations	
> ServiceID: 0x0F > This function is reentrant. > This function is synchronous. > This function is not allowed to be called in case that parallel service processing is enabled (see chapter *8.12.2 Diagnostic Client(s) Processing Parallelization* for more details).	

Table 5-26 Dcm_GetActiveProtocol()

5.2.2.2 Dcm_GetSecurityLevel()

Prototype	
`Std_ReturnType` **`Dcm_GetSecurityLevel`** `( Dcm_SecLevelType *SecLevel )`	
Parameter	
`SecLevel`	Active Security Level (see definition of Dcm_SecLevelType for values).
Return code	
`Std_ReturnType`	E_OK: This value is always returned.
Functional Description	
This function provides the active security level value. Note: According to [3]: Security level value = (<sub-function Id of requestSeed> + 1) / 2	
Particularities and Limitations	
> ServiceID: 0x0D > This function is reentrant. > This function is synchronous.	

Table 5-27 Dcm_GetSecurityLevel()

5.2.2.3 Dcm_GetSesCtrlType()

Prototype	
`Std_ReturnType` **`Dcm_GetSesCtrlType`** `( Dcm_SesCtrlType *SesCtrlType )`	
Parameter	
`SesCtrlType`	Active Session Control Type (see definition of Dcm_SesCtrlType for values).
Return code	
`Std_ReturnType`	E_OK: This value is always returned.
Functional Description	
This function provides the active session control type value.	
Particularities and Limitations	
> ServiceID: 0x06 > This function is reentrant. > This function is synchronous.	

Table 5-28 Dcm_GetSesCtrlType()

5.2.2.4 Dcm_ResetToDefaultSession()

Prototype	
`Std_ReturnType` **`Dcm_ResetToDefaultSession`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	E_OK: This value is always returned.
Functional Description	
The call to this function allows the application to reset the current session to Default session. Example: Automatic termination of an extended diagnostic session upon exceeding of a speed limit. Note: The time between the function call and the termination of the session depends on the current DCM state. The minimum time to be expected is one DCM task cycle. If this service is called while the DCM is processing a diagnostic request, the session termination will be postponed till the end of this service processing, to avoid unpredictable behavior.	
Particularities and Limitations	
> ServiceID: 0x2A > This function is reentrant. > This function is synchronous.	

Table 5-29 Dcm_ResetToDefaultSession()

5.2.2.5 Dcm_GetSecurityLevelFixedBytes()

Prototype	
`Std_ReturnType` **`Dcm_GetSecurityLevelFixedBytes`** `( Dcm_SecLevelType SecLevel, uint8 *FixedBytes, uint8 *BufferSize )`	
Parameter	
`SecLevel`	The security parameter, which fixed bytes are requested.
`FixedBytes`	Pointer to the buffer where the fixed bytes will be copied to.
`BufferSize`	IN: Specifies the available size of the provided buffer. OUT: Returns the number of copied fixed bytes, resp. number of required bytes to copy the complete set (in case of returned DCM_E_BUFFERTOOLOW).
Return code	
`Std_ReturnType`	E_OK: If the fixed bytes of the requested security level have been copied. For levels without fixed bytes, nothing will be copied, and the `BufferSize` parameter will be 0. DCM_E_BUFFERTOOLOW: If the level has fixed bytes, but the provided buffer is too small to fit them. The `BufferSize` will return the required buffer size. E_NOT_OK: If an invalid/unsupported security level or the “locked” level is passed to this API.
Functional Description	
By calling this API the application gets access to the fixed bytes set associated with the security-access level (i.e. any generated by the RTE DCM_SEC_LEV_XXX value) passed as selector. This API can be called at any time, but the most applicable situation is from within any of the *GetSeed()* or/and *CompareKey()* C/S callbacks. The implementation of the above callbacks shall be aware of passing the correct security-access level value that corresponds to its C/S port prototype. Otherwise the wrong values will be reported back.	
Particularities and Limitations	
> ServiceID: 0xA7 > This function is reentrant. > This function is synchronous. > Available only if at least one security level was configured to provide fixed bytes information.	

Table 5-30 Dcm_GetSecurityLevelFixedBytes()

5.2.2.6 Dcm_SetActiveDiagnostic()

Prototype	
`Std_ReturnType` **`Dcm_SetActiveDiagnostic`** `( boolean active )`	
Parameter	
`active`	Represents the type of DCM interaction with ComM: > TRUE: DCM shall call the *ComM_DCM_ActiveDiagnostic* > as required by [1]. > FALSE: DCM shall not call the *ComM_DCM_ActiveDiagnostic* > anymore.
Return code	
`Std_ReturnType`	E_OK: This code is always returned even if the action could not be executed due to: - Invalid value of `active`; - Not initialized DCM.
Functional Description	
This API shall be called by the application in cases where the sleep-prevention managed by DCM is no more desirable.	
Particularities and Limitations	
> ServiceID: 0x56 > This function is reentrant. > This function is synchronous.	

Table 5-31 Dcm_SetActiveDiagnostic()

5.2.2.7 Dcm_GetRequestKind()

Prototype	
`Std_ReturnType` **`Dcm_GetRequestKind`** `( uint16 TesterSourceAddress, Dcm_RequestKindType *RequestKind )`	
Parameter	
`TesterSourceAddress`	The source address of the tester which request kind status will be reported.
`RequestKind`	Returns the current request kind of the given diagnostic client: > DCM_REQ_KIND_NONE: Currently no request is in processing for this client. > DCM_REQ_KIND_EXTERNAL: An externally sent request for this client is in progress (i.e. reception/processing/transmission). > DCM_REQ_KIND_ROE: It is a STRT of RoE is in progress for this client.
Return code	
`Std_ReturnType`	E_OK: The `TesterSourceAddress` has a valid value. E_NOT_OK: An error occurred or the `TesterSourceAddress` has no valid value.
Functional Description	
This API can be called by the application at any time and from any context if information is required regarding the processing status of a certain diagnostic client. Typically this API can be used from within a *ServiceRequestManufacturerNotification_<SWC>* or *ServiceRequestSupplierNotification_<SWC>*, where the tester source address is passed as an argument, to get not only the request type (functional or physical) but also the kind of the request (internal/external). Additionally using the provided API *Dcm_GetTesterSourceAddress()*, the application may get the client request kind also from a known valid DcmRxPduId.	
Particularities and Limitations	
> ServiceID: 0xAB > This function is reentrant. > This function is synchronous. > This function may not be used with testers affiliated to generic connections.	

Table 5-32 Dcm_GetRequestKind()

5.2.2.8 Dcm_VsgSetSingle()

Prototype	
`Std_ReturnType` **`Dcm_VsgSetSingle`** `( Dcm_VsgIdentifierType VsgId, Dcm_VsgStateType State )`	
Parameter	
`VsgId`	Unique Identifier of a VSG (see *5.1.3 Dcm_VsgIdentifierType*).
`State`	New state to be set (for valid values see *5.1.4 Dcm_VsgStateType*).
Return code	
`Std_ReturnType`	E_OK: New state is set successfully. E_NOT_OK: New state is not set successfully.
Functional Description	
API to set the state of a single Vehicle System Group.	
Particularities and Limitations	
> ServiceID: 0xAC > This function is reentrant. > This function is synchronous.	

Table 5-33 Dcm_VsgSetSingle()

5.2.2.9 Dcm_VsgSetMultiple()

Prototype	
`Std_ReturnType` **`Dcm_VsgSetMultiple`** `( const Dcm_VsgIdentifierType *VsgIdList, uint16 VsgListSize, Dcm_VsgStateType State )`	
Parameter	
`VsgIdList`	Pointer to a list of VSG Identifiers (see *5.1.3 Dcm_VsgIdentifierType*).
`VsgListSize`	Number of VSG identifiers in `VsgIdList`.
`State`	New state to be set (for valid values see *5.1.4 Dcm_VsgStateType*).
Return code	
`Std_ReturnType`	E_OK: new state is set successfully for all VSG Identifiers in `VsgIdList`. E_NOT_OK: new state is not set successfully for all VSG Identifiers in `VsgIdList`. VSG identifiers that are set successfully remain in new state.
Functional Description	
API to set the state of a list of Vehicle System Groups.	
Particularities and Limitations	
> ServiceID: 0xAE > This function is reentrant. > This function is synchronous.	

Table 5-34 Dcm_VsgSetMultiple()

5.2.2.10 Dcm_VsgIsActive()

Prototype	
`Std_ReturnType` **`Dcm_VsgIsActive`** `( Dcm_VsgIdentifierType VsgId, Dcm_VsgStateType *State )`	
Parameter	
`VsgId`	Unique Identifier of a VSG (see *5.1.3 Dcm_VsgIdentifierType*).
`State`	Current state of the VSG if return code is E_OK (for valid values see *5.1.4 Dcm_VsgStateType*).
Return code	
`Std_ReturnType`	E_OK: Current state provided successfully. E_NOT_OK: Operation failed.
Functional Description	
API to query current state of a Vehicle Sytem Group.	
Particularities and Limitations	
> ServiceID: 0xAD > This function is reentrant. > This function is synchronous.	

Table 5-35 Dcm_VsgIsActive()

5.2.2.11 Dcm_VsgIsActiveAnyOf()

Prototype	
`Std_ReturnType` **`Dcm_VsgIsActiveAnyOf`** `( const Dcm_VsgIdentifierType *VsgIdList, uint16 VsgListSize, Dcm_VsgStateType *State )`	
Parameter	
`VsgIdList`	Pointer to a list of VSG Identifiers (see *5.1.3 Dcm_VsgIdentifierType*).
`VsgListSize`	Number of VSG identifiers in `VsgIdList`.
`State`	Result of operation if return code is E_OK (for valid values see 5.1.4 Dcm_VsgStateType).
Return code	
`Std_ReturnType`	E_OK: Operation succeeded, result provided in parameter `State`. E_NOT_OK: Operation failed.
Functional Description	
API to query if state of at least one Vehicle System Group in a list of Vehicle System Groups is enabled.	
Particularities and Limitations	
> ServiceID: 0xAF > This function is reentrant. > This function is synchronous.	

Table 5-36 Dcm_VsgIsActiveAnyOf()

5.2.2.12 Dcm_SetSpecificCauseCode()

Prototype	
`Std_ReturnType` **`Dcm_SetSpecificCauseCode`** `( Dcm_SpecificCauseCodeType SpecificCauseCode )`	
Parameter	
`SpecificCauseCode`	The specific cause code to be returned to the diagnostic client.
Return code	
`Std_ReturnType`	E_OK: Operation succeeded. E_NOT_OK: Operation failed.
Functional Description	
API to provide an additional byte (specific cause code) to the negative response of the request in progress. Only the first registered specific cause code will be stored. Any further call of this API within the same diagnostic request will have no effect. This API should be called only from within the Dcm_MainFunction(Worker) context.	
Particularities and Limitations	
> ServiceID: 0xB4 > This function is not reentrant. > This function is synchronous.	

Table 5-37 Dcm_SetSpecificCauseCode()

5.2.2.13 Dcm_SetSecurityBypass()

Prototype	
`Std_ReturnType` **`Dcm_SetSecurityBypass`** `( Boolean SecBypass )`	
Parameter	
`SecBypass`	Returns the current request kind of the given diagnostic client: > TRUE: DCM shall enable all security levels (except locked level) simultaneously (Bypass-Mode). > FALSE: DCM shall lock the ECU again.
Return code	
`Std_ReturnType`	E_OK: Operation succeeded. Bypass mode changed. E_NOT_OK: Operation failed. Possible reasons: > DCM is not yet initialized. > Wrong/invalid SecBypass parameter.
Functional Description	
The call to this function allows the application to bypass the internal DCM security states.	
Particularities and Limitations	
> ServiceID: 0xB3 > This function is not reentrant. > This function is asynchronous.	

Table 5-38 Dcm_SetSecurityBypass()

Caution
Using this function bypasses the ISO specific service 0x27 handling! Please see chapter *8.42 How to Bypass Pre-Condition Checks* for further details.

5.2.2.14 Dcm_SetAuthenticationBypass()

Prototype	
`Std_ReturnType` **`Dcm_SetAuthenticationBypass`** `( Boolean AuthBypass )`	
Parameter	
`SecBypass`	Returns the current request kind of the given diagnostic client: > TRUE: DCM shall bypass the authentication role checking prior to accessing diagnostic entities. > FALSE: DCM shall not bypass the authentication role checking.
Return code	
`Std_ReturnType`	E_OK: Operation succeeded. Bypass mode changed. E_NOT_OK: Operation failed. Possible reasons: > DCM is not yet initialized. > Wrong/invalid AuthBypass parameter.
Functional Description	
The call to this function allows the application to bypass the internal authentication role checking prior to accessing diagnostic entities.	
Particularities and Limitations	
> ServiceID: 0xBC > This function is not reentrant. > This function is asynchronous.	

Table 5-39 Dcm_SetAuthenticationBypass()

Caution
Using this function bypasses the authentication roles checking for services, DIDs, RIDs, memory ranges! Please see chapter *8.42 How to Bypass Pre-Condition Checks* for further details.

5.2.2.15 Dcm_SetDeauthenticatedRole()

Prototype	
`Std_ReturnType` **`Dcm_SetDeauthenticatedRole`** `( uint16 connectionId, const Dcm_AuthenticationRoleType deauthenticatedRole )`	
Parameter	
`connectionId`	The connection identifier (not the connection handle)
`deauthenticatedRole`	New deauthenticated role
Return code	
`Std_ReturnType`	E_OK: Operation succeeded. E_NOT_OK: Operation failed. Possible reasons: > DCM is not yet initialized. > Wrong/invalid parameters.
Functional Description	
Sets a new role used in deauthenticated state for that connection. The set role is valid until the connection switches into authenticated state or the ECU is reset. Note: The time between the function call and the update of the deauthenticated role depends on the current DCM state. The minimum time to be expected is one DCM task cycle. If this service is called while the DCM is processing a diagnostic request, the update will be postponed till the end of this service processing, to avoid unpredictable behavior. Nevertheless, the new role will be set before a new request for the same connectionId will be processed. Note: It is not allowed to call this interface with the same connectionId before a prior call to *Dcm_DeauthenticateConnection()* or *Dcm_AuthenticateConnection()* is realized.	
Particularities and Limitations	
> ServiceID: 0x79 > This function is reentrant for different connectionIds. > This function is asynchronous.	

Table 5-40 Dcm_SetDeauthenticatedRole()

5.2.3 General Purpose

5.2.3.1 Dcm_GetTesterSourceAddress()

Prototype	
`Std_ReturnType` **`Dcm_GetTesterSourceAddress`** `( PduIdType DcmRxPduId, uint16 *TesterSourceAddress )`	
Parameter	
`DcmRxPduId`	Specifies the DCM RxPduId for which the tester source address shall be read out.
`TesterSourceAddress`	Will contain the configured tester source address of the DCM RxPduId.
Return code	
`Std_ReturnType`	E_OK: The `TesterSourceAddress` has a valid value, or Dcm/DcmConfigSet/DcmGeneral/DcmDevErrorDetect is FALSE. E_NOT_OK: An error occurred, the `TesterSourceAddress` has no valid value, and Dcm/DcmConfigSet/DcmGeneral/DcmDevErrorDetect is TRUE.
Functional Description	
This API can be used to access the configured tester source address parameter to a specific DCM main-connection, identified by the DCM RxPduId. Usually this API is used in a project specific switch between software contexts (i.e. application and boot loader) where the request is received in one context (e.g. application) and the response is sent from the other context (e.g. boot loader) or vice versa.	
Particularities and Limitations	
> ServiceID: 0xA6 > This function is reentrant. > This function is synchronous. > This function may not be used with testers affiliated to generic connections.	

Table 5-41 Dcm_GetTesterSourceAddress()

5.2.3.2 Dcm_ProcessVirtualRequest()

Prototype	
`Std_ReturnType` **`Dcm_ProcessVirtualRequest`** `( PduIdType RxPduId, Dcm_MsgType Data, PduLengthType Length )`	
Parameter	
`RxPduId`	The DcmPduId (physical or functional) of the diagnostic client this virtual request represents. The response of this request will later be forwarded to this client.
`Data`	Pointer to the buffer where the complete diagnostic request incl. SID is located. In case of generic connections, the last byte should contain the source address of the client (tester).
`Length`	The length of the diagnostic request located in the `Data` buffer. Therefore, in case of generic connections, the last byte of the parameter Data (source address of the client) should not be considered in obtaining the parameter Length.
Return code	
`Std_ReturnType`	E_OK: The request has been accepted. E_NOT_OK: The request was not accepted. Possible reasons: > DCM is already busy with another client, resp. the RxPduId is from a low priority client > Invalid RxPduId, too long request or NULL_PTR for Data location passed to the API
Functional Description	
This is a generic API that can be used by the application (CDD) to send a virtual request to the ECU which response will be sent to a concrete diagnostic client. Typical use-case of this API is an application implementation for service 0x86 (ResponseOnEvent). This API can be called from any context (ISR, TASK, etc.). Just when called from an ISR and the request contains lots of data, the interrupt latency can be significantly affected.	
Particularities and Limitations	
> ServiceID: 0xA8 > This function is reentrant. > This function is synchronous.	

Table 5-42 Dcm_ProcessVirtualRequest()

5.2.3.3 Dcm_SetSecurityLevel()

Prototype	
`Std_ReturnType` **`Dcm_SetSecurityLevel`** `( Dcm_SecLevelType SecLevel )`	
Parameter	
`SecLevel`	Active Security Level (see definition of Dcm_SecLevelType for values).
Return code	
`Std_ReturnType`	E_OK: State change has been performed. E_NOT_OK: State change failed. Possible reasons: - Wrong/invalid security level; - Called while DCM is busy with a diagnostic request; - Called from wrong task context (not from Dcm_MainFunctionWorker);
Functional Description	
This API shall be called by the application when service *SecurityAccess (0x27)* is supported in the ECU but not handled in DCM. In this case DCM will be able to switch only to the LOCKED security level when performing a diagnostic session transition. To unlock the ECU in any other security level the application shall trigger the security access state transitions by calling this API with the appropriate value. Within this API call, DCM will perform the same RTE interaction as if the security state handling was done by itself. For that reason this API **must** be called only from within the Dcm_MainFunction(Worker) context. The best place for this call is either the callback function for *SecurityAccess (0x27)* or a *Confirmation()* if configured.	
Particularities and Limitations	
> ServiceID: 0xA9 > This function is not reentrant. > This function is synchronous. > Available only if service *SecurityAccess (0x27)* is supported in the ECU but handled within the DCM application.	

Table 5-43 Dcm_SetSecurityLevel()

5.2.3.4 Dcm_DemTriggerOnDTCStatus()

Prototype	
`Std_ReturnType` **`Dcm_DemTriggerOnDTCStatus`** `( uint32 DTC, uint8 DTCStatusOld, uint8 DTCStatusNew )`	
Parameter	
`DTC`	The DTC of which status byte just changed.
`DTCStatusOld`	The DTC status before the change (old status).
`DTCStatusNew`	The DTC status after change (current status).
Return code	
`Std_ReturnType`	E_OK: This value is always returned (even in error cases, e.g. DCM is not yet initialized).
Functional Description	
This API is called by DEM once DCM activates the DTC status change monitoring by calling the “**Dem_DcmControlDTCStatusChangedNotification()**” with parameter value **TRUE**. Once this API is called, DCM decides whether the DTC status change is relevant or not and triggers a corresponding action. Typically, this is RoE event.	
Particularities and Limitations	
> ServiceID: 0x2B > This function is reentrant. > This function is synchronous. > Available if service *ResponseOnEvent (0x86)* is supported in the ECU, event type onDTCStatusChange or reportDTCRecordInformationOnDtcStatusChange is supported and handled by DCM.	

Table 5-44 Dcm_DemTriggerOnDTCStatus()

5.2.3.5 Dcm_DeauthenticateConnection()

Prototype	
`Std_ReturnType` **`Dcm_DeauthenticateConnection`** `( uint16 connectionId )`	
Parameter	
`connectionId`	The connection identifier as configured.
Return code	
Std_ReturnType	E_OK: Operation succeeded. E_NOT_OK: Operation failed. Possible reasons: > DCM is not yet initialized. > Wrong/invalid parameters. > Recurrent call for the same connectionId.
Functional Description	
This API shall be called by the application when service 0x29 is fully implemented by the application, but the DCM inherent state checks shall be performed to allow authenticated access for testers. Or it shall be called when a tester needs to be deauthenticated on demand when service 0x29 is implemented internally. It sets the state of the given connectionId to deauthenticated and the role to the configured deauthenticated role. The state and role are valid until the connectionId is authenticated again. Note: The time between the function call and the deauthentication depends on the current DCM state. The minimum time to be expected is one DCM task cycle. If this service is called while the DCM is processing a diagnostic request for the same connectionId, the update will be postponed till the end of this service processing, to avoid unpredictable behavior. Nevertheless, the deauthentication will take place before a new request for the same connectionId will be processed. Note: It is not allowed to call this interface with the same connectionId before a prior call to *Dcm_SetDeauthenticatedRole() or Dcm_AuthenticateConnection()* is realized.	
Particularities and Limitations	
> ServiceID: 0xBA > This function is reentrant for different connectionIds. > This function is asynchronous. > The API is only available when service 0x29 is fully implemented externally or at least service 0x29 subservice 0x01 or 0x02 are used internally.	

Table 5-45 Dcm_DeauthenticateConnection()

5.2.3.6 Dcm_AuthenticateConnection()

Prototype	
`Std_ReturnType` **`Dcm_AuthenticateConnection`** `( uint16 connectionId, const Dcm_AuthenticationInfoType* authenticationInfo )`	
Parameter	
`connectionId`	The connection identifier as configured
`authenticationInfo`	Structure which holds the authenticated role and if configured the specific white list.
Return code	
Std_ReturnType	E_OK: Operation succeeded. E_NOT_OK: Operation failed. Possible reasons: > DCM is not yet initialized. > Wrong/invalid parameters. > Recurrent call for the same connectionId.
Functional Description	
This API shall be called by the application when service 0x29 is fully implemented by the application but the DCM inherent state checks shall be performed to allow authenticated access for testers. It sets the state of the given connectionId to authenticated with the given role and white list. The state/role/white list is valid until it is deauthenticated or updated by a new call. The white list is not copied directly to optimize memory consumption. During this call only the pointer to the white list is saved. The caller will be informed via the notification *Dcm_ConnectionAuthenticated()*, when the connection is authenticated. The data behind parameter authenticationInfo must be valid until this confirmation. Note: The time between the function call and the authentication depends on the current DCM state. The minimum time to be expected is one DCM task cycle. If this service is called while the DCM is processing a diagnostic request for the same connectionId, the update will be postponed till the end of this service processing, to avoid unpredictable behavior. Nevertheless, the authentication will take place before a new request for the same connectionId will be processed. Note: It is not allowed to call this interface with the same connectionId before a prior call to *Dcm_SetDeauthenticatedRole()* or *Dcm_DeauthenticateConnection()* is realized.	
Particularities and Limitations	
> ServiceID: 0xBB > This function is reentrant for different connectionIds. > This function is asynchronous. > The API is only available when service 0x29 is fully implemented externally.	

Table 5-46 Dcm_AuthenticateConnection()

5.2.3.7 Dcm_SetRoeActivationState()

Prototype	
`Std_ReturnType` **`Dcm_SetRoeActivationState`** `( boolean enabled )`	
Parameter	
`enabled`	State to be set: > True: RoE is active > False: RoE is inactive
Return code	
Std_ReturnType	E_OK: Operation succeeded. E_NOT_OK: Operation failed. Possible reasons: > DCM is not yet initialized. > Wrong/invalid parameter.
Functional Description	
This API shall be called by the application to dis-/enable the processing of RoE and sending of STRT. It sets the given state accordingly which suppresses STRT and its trigger when disabled.	
Particularities and Limitations	
> ServiceID: 0xBD > This function is not reentrant. > This function is synchronous.	

Table 5-47 Dcm_SetRoeActivationState()

5.3 Services used by DCM

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

Component	API
Dem	Dem_DcmCancelOperation Dem_[Dcm]EnableDTCRecordUpdate Dem_[Dcm]DisableDTCRecordUpdate Dem_[Dcm]SetFreezeFrameRecordFilter Dem_DcmGetOBDFreezeFrameData Dem_[Dcm]GetDTCStatusAvailabilityMask Dem_[Dcm]SetDTCFilter Dem_[Dcm]GetNextFilteredRecord Dem_[Dcm]GetNextFilteredDTCAndSeverity Dem_[Dcm]GetNextFilteredDTCAndFDC Dem_[Dcm]GetNextFilteredDTC Dem_[Dcm]GetExtendedDataRecordByDTC Dem_[Dcm]GetFreezeFrameDataByDTC Dem_[Dcm]GetNumberOfFilteredDTC Dem_[Dcm]GetSeverityOfDTC Dem_[Dcm]GetFunctionalUnitOfDTC Dem_[Dcm]GetStatusOfDTC Dem_[Dcm]GetTranslationType Dem_[Dcm]GetDTCByOccurrenceTime Dem_[Dcm]DisableDTCSetting Dem_[Dcm]EnableDTCSetting Dem_[Dcm]GetDTCOfOBDFreezeFrame Dem_[Dcm]ReadDataOfOBDFreezeFrame Dem_DcmControlDTCStatusChangedNotification Dem_DcmGetAvailableOBDMIDs Dem_DcmGetNumTIDsOfOBDMID Dem_DcmGetDTRData Dem_[Dcm]ClearDTC **For AUTOSAR Environment prior to 4.3.0** Dem_[Dcm]GetSizeOfExtendedDataRecordByDTC Dem_[Dcm]GetSizeOfFreezeFrameByDTC **For AUTOSAR Environment equal or newer than 4.3.0** Dem_SelectDTC Dem_GetDTCSelectionResult Dem_SelectExtendedDataRecord Dem_GetSizeOfExtendedDataRecordSelection Dem_GetNextExtendedDataRecord Dem_SelectFreezeFrameData

Component	API
	Dem_GetSizeOfFreezeFrameSelection Dem_GetNextFreezeFrameData Dem_SetExtendedDataRecordFilter Dem_GetSizeOfFilteredExtendedDataRecords Dem_GetNextFilteredExtendedDataRecord Dem_SetDTCFilterByReadinessGroup Dem_SetDTCFilterByExtendedDataRecordNumber
BswM	BswM_Dcm_ApplicationUpdated BswM_Dcm_CommunicationMode_CurrentState
Det	Det_ReportError
ComM	ComM_DCM_ActiveDiagnostic ComM_DCM_InactiveDiagnostic
PduR	PduR_DcmTransmit
SchM	SchM_Enter_Dcm_DCM_EXCLUSIVE_AREA_0 SchM_Exit_Dcm_DCM_EXCLUSIVE_AREA_0
NvM	NvM_ReadBlock NvM_WriteBlock NvM_CancelJobs NvM_SetBlockLockStatus NvM_GetErrorStatus NvM_GetDcmBlockId
EcuM	EcuM_BswErrorHook
KeyM	KeyM_SetCertificate KeyM_VerifyCertificate KeyM_GetCertificate KeyM_CertElementGet KeyM_CertElementGetFirst KeyM_CertElementGetNext
Csm	Csm_RandomGenerate Csm_SignatureGenerate Csm_SignatureVerify Csm_CancelJob

Table 5-48 Services used by the DCM

5.4 Callback Functions

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

5.4.1 <Module>

The following callbacks are to be used from the module that implements the callouts:

> *<Module>_<DiagnosticService>()*

> *<Module>_<DiagnosticService>_<SubService>()*

5.4.1.1 Dcm_ExternalProcessingDone()

Prototype	
`void` **`Dcm_ExternalProcessingDone`** `( Dcm_MsgContextType *pMsgContext )`	
Parameter	
`pMsgContext`	Message-related information for one diagnostic protocol identifier.
Return code	
`void`	N/A
Functional Description	
Used by service interpreter outside of DCM to indicate that the current diagnostic service processing is finished and (if required) a final response can be sent.	
Particularities and Limitations	
> ServiceID: 0x31 > This function is not reentrant. > This function is synchronous.	

Table 5-49 Dcm_ExternalProcessingDone()

5.4.1.2 Dcm_ExternalSetNegResponse()

Prototype	
`void` **`Dcm_ExternalSetNegResponse`** `( Dcm_MsgContextType *pMsgContext, Dcm_NegativeResponseCodeType ErrorCode )`	
Parameter	
`pMsgContext`	Message-related information for one diagnostic protocol identifier.
`ErrorCode`	Contains the NRC to be returned to the diagnostic client.
Return code	
`void`	N/A
Functional Description	
Used by service interpreter outside of DCM to indicate that a final response shall be a negative one. Dcm_ExternalSetNegResponse will not finalize the response processing.	
Particularities and Limitations	
> ServiceID: 0x30 > This function is not reentrant. > This function is synchronous.	

Table 5-50 Dcm_ExternalSetNegResponse()

5.4.2 ComM

5.4.2.1 Dcm_ComM_NoComModeEntered()

Prototype	
`void` **`Dcm_ComM_NoComModeEntered`** `( uint8 NetworkId )`	
Parameter	
`NetworkId`	Identifier of the network concerned by the mode change.
Return code	
`void`	N/A
Functional Description	
This call informs the DCM module about a ComM mode change to COMM_NO_COMMUNICATION.	
Particularities and Limitations	
> ServiceID: 0x21 > This function is reentrant. > This function is synchronous.	

Table 5-51 Dcm_ComM_NoComModeEntered()

5.4.2.2 Dcm_ComM_SilentComModeEntered()

Prototype	
`void` **`Dcm_ComM_SilentComModeEntered`** `( uint8 NetworkId )`	
Parameter	
`NetworkId`	Identifier of the network concerned by the mode change.
Return code	
`void`	N/A
Functional Description	
This call informs the DCM module about a ComM mode change to COMM_SILENT_COMMUNICATION.	
Particularities and Limitations	
> ServiceID: 0x22 > This function is reentrant. > This function is synchronous.	

Table 5-52 Dcm_ComM_SilentComModeEntered()

5.4.2.3 Dcm_ComM_FullComModeEntered()

Prototype	
`void` **`Dcm_ComM_FullComModeEntered`** `( uint8 NetworkId )`	
Parameter	
`NetworkId`	Identifier of the network concerned by the mode change.

Return code	
`void`	N/A
Functional Description	
This call informs the DCM module about a ComM mode change to COMM_FULL_COMMUNICATION.	
Particularities and Limitations	
> ServiceID: 0x23 > This function is reentrant. > This function is synchronous.	

Table 5-53 Dcm_ComM_FullComModeEntered()

5.4.2.4 Dcm_SetChannelReady()

Prototype	
`void` **`Dcm_SetChannelReady`** `( uint8 NetworkId )`	
Parameter	
`NetworkId`	Identifier of the network of which the availability changed.
Return code	
`void`	N/A
Functional Description	
This call informs the DCM module that a specific ComM channel is “Ready”. This means a TCP connection is established and routing enabled.	
Particularities and Limitations	
> ServiceID: 0xB7 > This function is reentrant. > This function is synchronous.	

Table 5-54 Dcm_SetChannelReady()

5.4.3 PduR

5.4.3.1 Dcm_TriggerTransmit()

Prototype	
`Std_ReturnType Dcm_TriggerTransmit ( PduIdType DcmTxPduId, PduInfoType *Info )`	
Parameter	
`DcmTxPduId`	ID of DCM I-PDU that has been transmitted. Range: 0..(maximum number of I-PDU IDs transmitted by DCM) - 1
`Info`	Pointer to the data buffer where to be transmitted data shall be copied to.
Return code	
`Std_ReturnType`	E_OK: If data has been copied. E_NOT_OK: In case of any error detected within this API.
Functional Description	
This is called by the PduR to get any data to be transmitted to a lower layer with timed triggered transmission (i.e. FlexRay).	
Particularities and Limitations	
> ServiceID: 0xA2 > This function is reentrant. > This function is synchronous.	

Table 5-55 Dcm_TriggerTransmit ()

5.4.3.2 Dcm_StartOfReception()

Prototype	
`BufReq_ReturnType Dcm_StartOfReception ( PduIdType DcmRxPduId, PduInfoType *Info, PduLengthType TpSduLength, PduLengthType *RxBufferSizePtr )`	
Parameter	
`DcmRxPduId`	Identifies the DCM data to be received. This information is used within the DCM to distinguish two or more receptions at the same time.
`Info`	Pointer to a structure containing content and length of the first frame or single frame including MetaData.
`TpSduLength`	This length identifies the overall number of bytes to be received.
`RxBufferSizePtr`	Length of the available buffer.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: The diagnostic request will be accepted. BUFREQ_E_NOT_OK: The diagnostic request will not be accepted at all (i.e. no free buffer or processing context). BUFREQ_E_OVFL: The diagnostic request could be accepted, but it will not fit the configured buffer and therefore is rejected.
Functional Description	
Called once to initialize the reception of a diagnostic request.	

Particularities and Limitations
> ServiceID: 0x00 > This function is reentrant. > This function is synchronous.

Table 5-56 Dcm_StartOfReception()

5.4.3.3 Dcm_CopyRxData()

Prototype	
`BufReq_ReturnType` **`Dcm_CopyRxData`** `( PduIdType DcmRxPduId, PduInfoType *PduInfoPtr, PduLengthType *RxBufferSizePtr )`	
Parameter	
`DcmRxPduId`	Identifies the DCM data to be received. This information is used within the DCM to distinguish two or more receptions at the same time.
`PduInfoPtr`	Pointer to a PduInfoType which indicates the number of bytes to be copied (SduLength) and the location of the source data (SduDataPtr). A SduLength of 0 is possible in order to poll the available receive buffer size. In this case no data are to be copied and PduInfoPtr might be invalid.
`RxBufferSizePtr`	Remaining free place in receive buffer after completion of this call.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: Data has been copied to the receive buffer completely as requested. BUFREQ_E_NOT_OK: Data has not been copied. Request failed.
Functional Description	
Called once upon reception of each segment. Within this call, the received data is copied from the receive TP buffer to the DCM receive buffer. The API might only be called with a SduLength greater 0 if the RxBufferSizePtr returned by the previous API call indicates sufficient receive buffer (SduLength <= RxBufferSizePtr). The function must only be called if the connection has been accepted by an initial call to Dcm_StartOfReception.	
Particularities and Limitations	
> ServiceID: 0x02 > Reentrant for different PduIds. Non-reentrant for the same PduId. > This function is synchronous.	

Table 5-57 Dcm_CopyRxData()

5.4.3.4 Dcm_TpRxIndication()

Prototype	
`void Dcm_TpRxIndication ( PduIdType DcmRxPduId, Std_ReturnType Result )`	
Parameter	
`DcmRxPduId`	ID of DCM I-PDU that has been received. Identifies the data that has been received. Range: 0..(maximum number of I-PDU IDs received by DCM) – 1
`Result`	E_OK: The complete N-PDU has been received and is stored in the receive buffer E_NOT_OK: The N_PDU has not been received properly, DCM should prepare for a new reception.
Return code	
`void`	N/A
Functional Description	
This is called by the PduR to indicate the completion of a reception.	
Particularities and Limitations	
> ServiceID: 0x03 > This function is reentrant. > This function is synchronous.	

Table 5-58 Dcm_TpRxIndication()

5.4.3.5 Dcm_CopyTxData()

Prototype	
`BufReq_ReturnType` **`Dcm_CopyTxData`** `( PduIdType DcmTxPduId, PduInfoType *PduInfoPtr, RetryInfoType *RetryInfoPtr, PduLengthType *TxDataCntPtr )`	
Parameter	
`DcmTxPduId`	Identifies the DCM data to be sent. This information is used to derive the PCI information within the transport protocol. The value must be same as in the according service call PduR_DcmTransmit().
`PduInfoPtr`	Pointer to a PduInfoType, which indicates the number of bytes to be copied (SduLength) and the location where the data must be copied to (SduDataPtr). An SduLength of 0 is possible in order to poll the available transmit data count. In this case no data are to be copied and SduDataPtr might be invalid.
`RetryInfoPtr`	If the transmitted TP I-PDU does not support the retry feature a NULL_PTR can be provided. This indicates that the copied transmit data can be removed from the buffer after it has been copied.
`TxDataCntPtr`	Remaining Tx data after completion of this call.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: Data has been copied to the transmit buffer completely as requested. BUFREQ_E_NOT_OK: Data has not been copied. Request failed, in case the corresponding I-PDU was stopped. BUFREQ_E_BUSY: There is temporarily not enough data to be transmitted. Retry later.
Functional Description	
At invocation of Dcm_CopyTxData the DCM module copies the requested transmit data with ID PduId from its internal transmit buffer to the location specified by the PduInfoPtr. The function Dcm_CopyTxData also calculates and sets the TxDataCntPtr to the number of remaining bytes for the transmission of this data. If RetryInfoPtr is NULL_PTR or if TpDataState is equal to TP_DATACONF, the DCM shall always copy the next fragment of data to the SduDataPtr. No TpDataState other than TP_DATACONF is supported by the current DCM implementation.	
Particularities and Limitations	
> ServiceID: 0x04 > Reentrant for different PduIds. Non-reentrant for the same PduId. > This function is synchronous.	

Table 5-59 Dcm_CopyTxData()

5.4.3.6 Dcm_TpTxConfirmation()

Prototype	
`void Dcm_TpTxConfirmation ( PduIdType DcmTxPduId, Std_ReturnType Result )`	
Parameter	
`DcmTxPduId`	ID of DCM I-PDU that has been transmitted. Range: 0..(maximum number of I-PDU IDs transmitted by DCM) – 1
`Result`	E_OK: The complete N-PDU has been transmitted. E_NOT_OK: An error occurred during transmission, the DCM can unlock the transmit buffer.
Return code	
`void`	N/A
Functional Description	
This is called by the PduR to confirm an end of transport protocol (e.g. CanTp) transmission.	
Particularities and Limitations	
> ServiceID: 0x05 > This function is reentrant. > This function is synchronous.	

Table 5-60 Dcm_TpTxConfirmation()

5.4.3.7 Dcm_TxConfirmation()

Prototype	
`void Dcm_TxConfirmation ( PduIdType DcmTxPduId )`	
Parameter	
`DcmTxPduId`	ID of DCM I-PDU that has been transmitted. Range: 0..(maximum number of I-PDU IDs transmitted by DCM) – 1
Return code	
`void`	N/A
Functional Description	
This is called by the PduR to confirm an end of interface (e.g. CanIf) transmission.	
Particularities and Limitations	
> ServiceID: 0xA1 > This function is reentrant. > This function is synchronous.	

Table 5-61 Dcm_TxConfirmation()

5.4.4 CanTp

5.4.4.1 Dcm_OnRequestDetection()

Prototype	
`void Dcm_OnRequestDetection ( PduIdType CanTpRxPduId, uint8 TpAddrExtension )`	
Parameter	
`CanTpRxPduId`	Represents the CanIf to CanTp RxPduId of the request.
`TpAddrExtension`	Defines the address extension byte value of the message.
Return code	
`void`	N/A
Functional Description	
This API will be called by the CanTp each time a new TP CAN frame of type first-frame or single-frame is received. The DCM will check whether this CAN message applies to any DCM connection (i.e. the CAN message is one of the DCM clients' physical requests). If so, any ongoing diagnostic (request/response) transmission over this client connection will be terminated. Additionally, if there is service processing in progress, it will be terminated too.	
Particularities and Limitations	
> ServiceID: 0xA4 > This function is reentrant. > This function is synchronous. > This function is only available if DCM shall support Mixed11 addressing CanTp connections.	

Table 5-62 Dcm_ OnRequestDetection()

5.4.5 KeyM

5.4.5.1 Dcm_KeyMAsyncCertificateVerifyFinished()

Prototype	
`Std_ReturnType` **`Dcm_KeyMAsyncCertificateVerifyFinished`** `( KeyM_CertificateIdType CertId, KeyM_CertificateStatusType Result )`	
Parameter	
`KeyM_CertificateIdType`	The certificate identifier that has been verified.
`KeyM_CertificateStatusType`	Contains information about the result of the operation.
Return code	
`E_OK`	This value is always returned.
Functional Description	
Called by Key Manager when finished with a certificate verify job.	
Particularities and Limitations	
> ServiceID = 0xB1 > This function is not reentrant. > This function is synchronous. > This function is only available if DCM supports Authentication Manager.	

Table 5-63 Dcm_KeyMAsyncCertificateVerifyFinished()

5.4.6 Csm

5.4.6.1 Dcm_CsmAsyncJobFinished()

Prototype	
`void` **`Dcm_CsmAsyncJobFinished`** `( const Crypto_JobType* job, Crypto_ResultType result )`	
Parameter	
`job`	Job of the operation that caused the callback.
`result`	Contains the result of the Csm operation.
Return code	
`void`	N/A
Functional Description	
Called by Crypto Service Manager when finished with a job.	
Particularities and Limitations	
> ServiceID = 0xB2 > This function is not reentrant. > This function is synchronous. > This function is only available if DCM supports Authentication Manager.	

Table 5-64 Dcm_CsmAsyncJobFinished()

5.4.6.2 Dcm_CsmSecureCodingValidationFinished()

Prototype	
`void` **`Dcm_CsmSecureCodingValidationFinished`** `( const Crypto_JobType *job, Crypto_ResultType result )`	
Parameter	
`job`	Job of the operation that caused the callback.
`result`	Contains the result of the Csm operation.
Return code	
`void`	N/A
Functional Description	
Called by Crypto Service Manager when a verification job for the Secure Coding feature is finished.	
Particularities and Limitations	
> ServiceID: 0xB5 > This function is not reentrant. > This function is synchronous. > This function is only available when DCM supports Secure Coding feature.	

Table 5-65 Dcm_CsmSecureCodingValidationFinished()

5.5 Configurable Interfaces

5.5.1 Notifications

5.5.1.1 <Diagnostic Session Change Notification Callback>

Prototype	
`void <Diagnostic Session Change Callback> ( Dcm_SesCtrlType FormerSesCtrlId, Dcm_SesCtrlType NewSesCtrlId )`	
Parameter	
`FormerSesCtrlId`	Specifies the former diagnostic session ID (transition's source state)
`NewSesCtrlId`	Specifies the new diagnostic session ID (transition's target state)
Return code	
–	-
Functional Description	
Any configured function of this kind will be called at a diagnostic session state transition. **Note:** The function argument values have the same definition as the ones returned by the API *Dcm_GetSesCtrlType()*. Please refer also to *8.17 How to Know When the Diagnostic Session Changes* for more details on how to configure such a callback and when it will be called.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	

Table 5-66 < Diagnostic Session Change Notification Callback >

5.5.1.2 <Security Access Change Notification Callback>

Prototype	
`void <Security Access Change Callback> ( Dcm_SecLevelType FormerSecLevelId, Dcm_SecLevelType NewSecLevelId )`	
Parameter	
`FormerSecLevelId`	Specifies the former security access level ID (transition's source state)
`NewSecLevelId`	Specifies the new security access level ID (transition's target state)
Return code	
-	-
Functional Description	
Any configured function of this kind will be called at a security access level state transition. **Note:** The function argument values have the same definition as the ones returned by the API *Dcm_GetSecurityLevel()*. Please refer also to *8.18.2 Calling a Function Implemented Within a CDD Module* for more details on how to configure such a callback and when it will be called.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	

Table 5-67 <Security Access Change Notification Callback>

5.5.1.3 Dcm_ConnectionAuthenticated()

Prototype	
`void Dcm_ConnectionAuthenticated ( uint16 connectionId )`	
Parameter	
`connectionId`	The connection identifier as configured
Return code	
-	-
Functional Description	
When service 0x29 is implemented externally and a connection is authenticated via *Dcm_AuthenticateConnection()*, the caller is notified with this notification.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > This notification will be called only when service 0x29 is fully implemented externally.	

Table 5-68 Dcm_ConnectionAuthenticated()

5.5.2 Callout Functions

At its configurable interfaces, the DCM defines callout functions. The declarations of the callout functions are provided by the BSW module, i.e. the DCM. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The DCM callout function declarations are described in the following tables:

5.5.2.1 <Module>_<DiagnosticService>()

Prototype	
AUTOSAR Version older than 4.2.2	
`Std_ReturnType` **`<Module>_<DiagnosticService>`** `( Dcm_OpStatusType OpStatus, Dcm_MsgContextType *pMsgContext )`	
AUTOSAR Version equal or newer than 4.2.2 and older than R19-11	
`Std_ReturnType` **`<Module>_<DiagnosticService>`** `( Dcm_OpStatusType OpStatus, Dcm_MsgContextType *pMsgContext, Dcm_NegativeResponseCodeType *ErrorCode )`	
AUTOSAR Version equal or newer than R19-11	
`Std_ReturnType` **`<Module>_<DiagnosticService>`** `( Dcm_ExtendedOpStatusType OpStatus, Dcm_MsgContextType *pMsgContext, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`pMsgContext`	Request/Response message-related information.
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only. DCM_FORCE_RCRRP_OK: (Vendor extension) The enforced RCR-RP transmission has finished with success. DCM_FORCE_RCRRP_NOT_OK: (Vendor extension) The enforced RCR-RP transmission has failed
	Additional for AUTOSAR Version equal or newer than R19-11
	DCM_POS_RESPONSE_SENT: Not used. DCM_POS_RESPONSE_FAILED: Not used. DCM_NEG_RESPONSE_SENT: Not used. DCM_NEG_RESPONSE_FAILED: Not used.
`ErrorCode`	Returns the NRC to be sent in the negative response.
Return code	
AUTOSAR Version equal or newer than 4.2.2	
`Std_ReturnType`	E_OK: Job processing finished, send positive response. E_NOT_OK: Job processing finished, send NRC from the ErrorCode. DCM_E_PENDING: Job processing is not yet finished. DCM_E_FORCE_RCRRP: (Vendor extension) Forces an RCR-RP response. The call out will called again once the response is sent. The OpStatus

	parameter will contain the transmission result.
AUTOSAR Version older than 4.2.2	
`Std_ReturnType`	E_OK: Job processing finished, send positive response. DCM_E_PENDING: Job processing is not yet finished. DCM_E_FORCE_RCRRP: (Vendor extension) Forces a RCR-RP response. The call out will called again once the response is sent. The OpStatus parameter will contain the transmission result. DCM_E_PROCESSINGDONE: (Vendor extension): Can be returned instead of calling Dcm_ExternalProcessingDone() for the current pMsgContext. Saves application code and stack usage.

Functional Description

AUTOSAR Version equal or newer than 4.2.2

DCM calls a function of this kind as soon as a supported diagnostic service, configured to be handled by a CDD, is received. All the relevant diagnostic request parameter information is forwarded by DCM through the pMsgContext function parameter.

The concrete name of the callout is defined by the configuration parameter /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSidTabFnc.

Description of pMsgContext parameter values, set by DCM for diagnostic service level processing (all other members are specified in [1] and within the definition of **Dcm_MsgContextType**):

`pMsgContext->reqDataLen` - contains the total request length less one (SID) byte.

`pMsgContext->resDataLen` - is set to 0. If any data shall be returned in the positive response (i.e. something written at `pMsgContext->resData`), return its size through this member.

`pMsgContext->resMaxDataLen` - contains the available response buffer size.

`pMsgContext->reqData` - points to the request data behind the SID byte.

`pMsgContext->resData` - points to the response data behind the SID byte.

Job processing finished:

> With success: Commit any applicable data size in `pMsgContext->resDataLen` and return E_OK.

With failure: Put an appropriate NRC to `ErrorCode` and return E_NOT_OK.

AUTOSAR Version older than 4.2.2

Refer to AUTOSAR Version equal or newer than 4.2.2 except:

Job processing finished:

> With success: Commit any applicable data size in `pMsgContext->resDataLen`, call *Dcm_ExternalProcessingDone()* and return E_OK.

With failure: Call *Dcm_ExternalSetNegResponse()* with appropriate NRC and finish as in “with success”.

Particularities and Limitations

> ServiceID: 0x32
> This function is reentrant.
> This function is asynchronous.

Table 5-69 <Module>_<DiagnosticService>()

5.5.2.2 <Module>_<DiagnosticService>_<SubService>()

Prototype
AUTOSAR Version older than 4.2.2
`Std_ReturnType` **`<Module>_<DiagnosticService>_<SubService>`** `( Dcm_OpStatusType OpStatus, Dcm_MsgContextType *pMsgContext )`
AUTOSAR Version equal or newer than 4.2.2 and older than R19-11
`Std_ReturnType` **`<Module>_<DiagnosticService>_<SubService>`** `( Dcm_OpStatusType OpStatus, Dcm_MsgContextType *pMsgContext, Dcm_NegativeResponseCodeType *ErrorCode )`
AUTOSAR Version equal or newer than R19-11
`Std_ReturnType` **`<Module>_<DiagnosticService>_<SubService>`** `( Dcm_ExtendedOpStatusType OpStatus, Dcm_MsgContextType *pMsgContext, Dcm_NegativeResponseCodeType *ErrorCode )`

Parameter	
`pMsgContext`	Request/Response message-related information.
`OpStatus`	Refer to *5.5.2.1 <Module>_<DiagnosticService>()*
`ErrorCode`	Returns the NRC to be sent in the negative response.

Return code	
`Std_ReturnType`	Refer to *5.5.2.1 <Module>_<DiagnosticService>()*

Functional Description

AUTOSAR Version equal or newer than 4.2.2

DCM calls a function of this kind as soon as a supported diagnostic sub-service, configured to be handled by a CDD, is received. All of the relevant diagnostic request parameter information is forwarded by DCM through the pMsgContext function parameter.

The concrete name of the callout is defined by the configuration parameter /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService/DcmDsdSubServiceFnc.

Description of pMsgContext parameter values, set by DCM for diagnostic sub-service level processing (all other members are specified in [1] and within the definition of **Dcm_MsgContextType**):

`pMsgContext->reqDataLen` - contains the total request length less two (SID + SF) bytes.

`pMsgContext->resDataLen` - is set to 0. If any data shall be returned in the positive response (i.e. something written at `pMsgContext->resData`), return its size through this member.

`pMsgContext->resMaxDataLen` - contains the available response buffer size.

`pMsgContext->reqData` - points to the request data behind the SF byte.

`pMsgContext->resData` - points to the response data behind the SF byte.

Job processing finished:

> With success: Commit any applicable data size in `pMsgContext->resDataLen` and return E_OK.

With failure: Put an appropriate NRC to `ErrorCode` and return E_NOT_OK.

AUTOSAR Version older than 4.2.2
Refer to AUTOSAR Version equal or newer than 4.2.2 except: Job processing finished: > With success: Commit any applicable data size in `pMsgContext->resDataLen`, call *Dcm_ExternalProcessingDone()* and return E_OK. With failure: Call *Dcm_ExternalSetNegResponse()* with appropriate NRC and finish as in “with success”.
Particularities and Limitations
> ServiceID: 0x33 > This function is reentrant. > This function is asynchronous.

Table 5-70 <Module>_<DiagnosticService>_<SubService>()

5.5.2.3 <Module>_<DiagnosticService>_<PreHandler>()

Prototype	
`Std_ReturnType` **`<Module>_<DiagnosticService>_<PreHandler>`** `(Dcm_OpStatusType OpStatus, uint8 SID, const uint8 *RequestData, uint32 DataSize, Dcm_ProtocolType ProtocolType, uint16 TesterSourceAddress, uint16 ConnectionId, uint8 ReqType, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only.
`SID`	The diagnostic service Id.
`RequestData`	Points to the request data. Points behind the service Id byte.
`DataSize`	The requested data length (without the SID byte).
`ProtocolType`	The ID of the protocol on which the request was received.
`TesterSourceAddress`	The diagnostic client source address.
`ConnectionId`	Unique ID of the connection on which the request has been received.
`ReqType`	The type of the diagnostic request: 0 - physical request, 1 - functional request.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK). A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Return code	
`Std_ReturnType`	E_OK: Service processing can be started. DCM_E_PENDING: The application needs more time to evaluate. E_NOT_OK: Service processing shall be denied. Send NRC.
Functional Description	
The <Module>_<DiagnosticService>_<PreHandler> callout is called before the actual diagnostic service is processed to enable the application to fulfill preconditions or alter internal, needed states.	

Particularities and Limitations
> ServiceID: 0xB8 > This function is not reentrant. > This function is asynchronous.

Table 5-71 <Module>_<DiagnosticService>_<PreHandler>()

5.5.2.4 <Module>_<DiagnosticService>_<PostHandler>()

Prototype
`Std_ReturnType` **`<Module>_<DiagnosticService>_<PostHandler>`** `(Dcm_OpStatusType OpStatus, uint8 SID, const uint8 *ResponseData, uint32 DataSize, Dcm_ProtocolType ProtocolType, uint16 TesterSourceAddress, uint16 ConnectionId, uint8 ReqType, Dcm_NegativeResponseCodeType *ErrorCode )`

Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only.
`SID`	The diagnostic service Id.
`ResponseData`	Points to the response data. Points behind the service Id byte.
`DataSize`	The requested data length (without the SID byte).
`ProtocolType`	The ID of the protocol on which the request was received.
`TesterSourceAddress`	The diagnostic client source address.
`ConnectionId`	Unique ID of the connection on which the request has been received.
`ReqType`	The type of the diagnostic request: 0 - physical request, 1 - functional request.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK). A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Return code	
`Std_ReturnType`	E_OK: CleanUp successful. DCM_E_PENDING: The application needs more time to evaluate. E_NOT_OK: CallOut not successful. Send NRC.

Functional Description
The <Module>_<DiagnosticService>_<PostHandler> callout is called after the actual diagnostic service is processed to enable the application to clean up if necessary or be informed of result of service processing.

Particularities and Limitations
> ServiceID: 0xB9 > This function is not reentrant. > This function is asynchronous.

Table 5-72 <Module>_<DiagnosticService>_<PostHandler>()

5.5.2.5 Dcm_SetProgConditions()

Prototype	
`Std_ReturnType` **`Dcm_SetProgConditions`** `( [Dcm_OpStatusType OpStatus, ] Dcm_ProgConditionsType *ProgConditions )`	
Parameter	
`OpStatus`	**Optional parameter.** Exists only in AR R19-11 or later enabled DCMs. DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only.
`ProgConditions`	Conditions on which the jump to bootloader or ECU reset has been requested.
Return code	
`Std_ReturnType`	E_OK: Conditions have correctly been set. E_NOT_OK: Conditions cannot be set. DCM_E_PENDING: Conditions set is in progress, a further call to this API is needed to end the setting.
Functional Description	
The Dcm_SetProgConditions callout allows the integrator to store relevant information prior to jumping to bootloader, or starting the ECU reset execution when positive response is configured to be sent afterwards. The context parameters are defined in Dcm_ProgConditionsType.	
Particularities and Limitations	
> ServiceID: 0x61 > This function is not reentrant. > This function is asynchronous.	

Table 5-73 Dcm_SetProgConditions()

5.5.2.6 Dcm_GetProgConditions()

Prototype	
`Dcm_EcuStartModeType` **`Dcm_GetProgConditions`** `( Dcm_ProgConditionsType *ProgConditions )`	
Parameter	
`ProgConditions`	Conditions on which the jump from the bootloader or ECU reset has been requested.
Return code	
`Dcm_EcuStartModeType`	DCM_COLD_START: The ECU starts normally. DCM_WARM_START: The ECU starts from a bootloader jump, or from ECU reset when positive response is configured to be sent after the reset execution. The function parameter values will be evaluated for further processing.
Functional Description	
The Dcm_GetProgConditions callout is called upon DCM initialization and allows determining if a response (0x50 or 0x51) must be sent depending on request within the bootloader or before the ECU reset execution. The context parameters are defined in Dcm_ProgConditionsType.	
Particularities and Limitations	
> ServiceID: 0x62 > This function is not reentrant. > This function is synchronous.	

Table 5-74 Dcm_GetProgConditions()

5.5.2.7 Dcm_Confirmation()

Prototype	
`void Dcm_Confirmation ( Dcm_IdContextType IdContext, PduIdType DcmRxPduId, Dcm_ConfirmationStatusType Status )`	
Parameter	
`IdContext`	Current context identifier which can be used to retrieve the relation between request and confirmation. Within the confirmation, the Dcm_MsgContext is no more available, so the IdContext can be used to represent this relation. The IdContext is also part of the Dcm_MsgContext.
`DcmRxPduId`	DcmRxPduId on which the request was received. The source of the request can have consequences for message processing.
`Status`	Status indication about confirmation (differentiate failure indication and normal confirmation) / The parameter "Result" of "Dcm_TxConfirmation" shall be forwarded to Status depending if a positive or negative response was sent before.
Return code	
`void`	N/A
Functional Description	
This function confirms the successful transmission or a transmission error of a diagnostic service. The IdContext and the DcmRxPduId are required to identify the message which was processed. If there was no response for this request, this call out is invoked at service processing finish. Note: This call out is invoked only then when a DCM internal or external <Module>_<DiagnosticService> service handler has been executed.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	

Table 5-75 Dcm_Confirmation()

5.5.2.8 Dcm_ReadMemory()

Prototype	
`Dcm_ReturnReadMemoryType` **`Dcm_ReadMemory`** `( Dcm_OpStatusType OpStatus, uint8 MemoryIdentifier, uint32 MemoryAddress, uint32 MemorySize, uint8 *MemoryData [, Dcm_NegativeResponseCodeType *ErrorCode] )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only. DCM_FORCE_RCRRP_OK: (Vendor extension) The enforced RCR-RP transmission has finished with success. DCM_FORCE_RCRRP_NOT_OK: (Vendor extension) The enforced RCR-RP transmission has failed.
`MemoryIdentifier`	MemoryIdentifier Identifier of the Memory Block (e.g. used if memory section distinguishing is needed). Note: If it is not used this parameter shall be set to 0.
`MemoryAddress`	Starting address of server memory from which data is to be retrieved.
`MemorySize`	Number of bytes in the MemoryData.
`MemoryData`	Data read (Points to the diagnostic buffer in DCM).
`ErrorCode`	**Optional parameter.** Exists only in AR 4.2.2 or later enabled DCMs. If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case DCM_READ_FAILED is returned.
Return code	
`Dcm_ReturnReadMemoryType`	DCM_READ_OK: Read was successful. DCM_READ_FAILED: Read was not successful. DCM_READ_PENDING: Read is not yet finished. DCM_READ_FORCE_RCRRP: Enforce RCR-RP transmission (vendor extension).
Functional Description	
The Dcm_ReadMemory callout is used to request memory data identified by the parameter memoryAddress and memorySize from the UDS request message. This service is needed for the implementation of UDS services: > ReadMemoryByAddress (0x23) > ReadDataByIdentifier (0x22) (in case of Dynamical DID defined by memory address) > ReadDataByPeriodicIdentifier (0x2A) (in case of Dynamical DID defined by memory address)	
Particularities and Limitations	
> ServiceID: 0x26 > This function is not reentrant. > This function is asynchronous.	

Table 5-76 Dcm_ReadMemory()

5.5.2.9 Dcm_WriteMemory()

Prototype	
`Dcm_ReturnWriteMemoryType` **`Dcm_WriteMemory`** `( Dcm_OpStatusType OpStatus, uint8 MemoryIdentifier, uint32 MemoryAddress, uint32 MemorySize, uint8 *MemoryData[, Dcm_NegativeResponseCodeType *ErrorCode] )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only. DCM_FORCE_RCRRP_OK: (Vendor extension) The enforced RCR-RP transmission has finished with success. DCM_FORCE_RCRRP_NOT_OK: (Vendor extension) The enforced RCR-RP transmission has failed.
`MemoryIdentifier`	MemoryIdentifier Identifier of the Memory Block (e.g. used if memory section distinguishing is needed). Note: If it is not used this parameter shall be set to 0.
`MemoryAddress`	Starting address of server memory where the data is to be written.
`MemorySize`	Number of bytes in the MemoryData.
`MemoryData`	Data to be written (Points to the diagnostic buffer in DCM).
`ErrorCode`	**Optional parameter.** Exists only in AR 4.2.2 or later enabled DCMs. If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case DCM_WRITE_FAILED is returned.
Return code	
`Dcm_ReturnWriteMemoryType`	DCM_WRITE_OK: Write was successful. DCM_WRITE_FAILED: Write was not successful. DCM_WRITE_PENDING: Write is not yet finished. DCM_WRITE_FORCE_RCRRP: Enforce RCR-RP transmission (vendor extension).
Functional Description	
The Dcm_WriteMemory callout is used to write memory data identified by the parameter memoryAddress and memorySize. This API is needed for the implementation of UDS services: > WriteMemoryByAddress (0x3D)	
Particularities and Limitations	
> ServiceID: 0x27 > This function is not reentrant. > This function is asynchronous.	

Table 5-77 Dcm_WriteMemory()

5.5.2.10 Dcm_GetRecoveryStates()

Prototype	
`Std_ReturnType Dcm_GetRecoveryStates ( Dcm_RecoveryInfoType *RecoveryInfo )`	
Parameter	
`RecoveryInfo`	Contains all the information that must be recovered.
Return code	
`Std_ReturnType`	E_OK: Recovery info is available and valid, process it. DCM_E_PENDING: Recovery info not yet available, call again. E_NOT_OK: No information to be recovered or result reading failed. DCM will continue with the default initialized states.
Functional Description	
This API will be called by DCM within the first *Dcm_MainFunction()* call right after the call of *Dcm_Init()*. For details on the usage of this API, please refer chapter *8.27 How to Recover DCM State Context on ECU Reset/Power On*. Note: > If no recovery of any state is needed (default startup of DCM), then the return value shall always be E_NOT_OK. > Before this API is called, DCM will lock any external connections until the result is processed. This is required to be able to switch into a consistent state without any influence from outside. > For details on the recovered information, please refer the data type definition: *Dcm_RecoveryInfoType*.	
Particularities and Limitations	
> ServiceID: 0xA5 > This function is not reentrant. > This function is asynchronous.	

Table 5-78 Dcm_GetRecoveryStates()

Caution
It is not intended to use *Dcm_GetRecoveryStates()* as a standalone API. The data it transfers depends on the DCM implementation version. For optimization reasons, the data structure uses internal data representation and not any official DCM AR APIs (e.g. macros for session and security access, or communication channel SNVs). Thus, it shall be used only if the information provider (*Dcm_ProvideRecoveryStates()*) has been used.

5.5.2.11 Dcm_FilterDidLookUpResult()

Prototype	
`Std_ReturnType Dcm_FilterDidLookUpResult ( Dcm_OpStatusType OpStatus, uint16 Did, Dcm_DidOpType DidOperation )`	
Parameter	
`OpStatus`	Status of the current operation.
`Did`	Data Identifier to be filtered.
`DidOperation`	DCM_DID_OP_READ: Available for services 0x22, 0x2A. DCM_DID_OP_WRITE: Available for service 0x2E. DCM_DID_OP_IO: Available for service 0x2F. DCM_DID_OP_SCALINGINFO: Available for service 0x24. DCM_DID_OP_DEFINE: Available for service 0x2C.
Return code	
`Std_ReturnType`	E_OK: The DID is (still) active. DCM_E_PENDING: The DID validation needs more time. Call this API again. E_NOT_OK: The DID is not active.
Functional Description	
This API will be called by DCM to check whether a combination of a DID and a DID operation is still supported. The return of that API is E_OK if that DID is active for the provided DID operation. This API is used in the filtering feature described in *8.29 How to Handle Multiple Diagnostic Service Variants*.	
Particularities and Limitations	
> This function is not reentrant. > This function is asynchronous.	

Table 5-79 Dcm_FilterDidLookUpResult()

5.5.2.12 Dcm_FilterRidLookUpResult()

Prototype	
`Std_ReturnType` **`Dcm_FilterRidLookUpResult`** `( Dcm_OpStatusType OpStatus, uint16 Rid )`	
Parameter	
`OpStatus`	Status of the current operation.
`Rid`	Routine Identifier to be filtered.
Return code	
`Std_ReturnType`	E_OK: The RID is (still) active. DCM_E_PENDING: The RID validation needs more time. Call this API again. E_NOT_OK: The RID is not active.
Functional Description	
This API will be called by DCM to check whether a RID is still supported. The return of that API is E_OK if that RID is active. This API is used in the filtering feature illustrated in *8.29 How to Handle Multiple Diagnostic Service Variants*.	
Particularities and Limitations	
> This function is not reentrant. > This function is asynchronous.	

Table 5-80 Dcm_FilterRidLookUpResult()

5.5.2.13 Dcm_HandleServiceExtern()

Prototype	
`boolean` **`Dcm_HandleServiceExtern`** `( uint8 SID, const uint8 *RequestData, uint16 DataSize, uint8 ReqType, uint16 SourceAddress )`	
Parameter	
`SID`	Contains the diagnostic service Id.
`RequestData`	Points to the request data. Points behind the service Id byte.
`DataSize`	Specifies the requested data length (without the SID byte).
`ReqType`	Specifies the diagnostic request type: 0 - physical request, 1 - functional request.
`SourceAddress`	Contains the diagnostic client source address.
Return code	
`boolean`	TRUE: The service processing is done by the application. FALSE: The service processing is done by DCM.
Functional Description	
The Dcm_HandleServiceExtern call-out is called during request processing if the service is configured to be dispatchable. This API is used in the diagnostic service dispatching feature illustrated in *8.36 How to Support Diagnostic Service Dispatching*.	

Particularities and Limitations
> ServiceID: 0xB0 > This function is not reentrant. > This function is synchronous.

Table 5-81 Dcm_HandleServiceExtern()

5.5.2.14 Dcm_GetAuthenticationRoles()

Prototype	
`Std_ReturnType` **`Dcm_GetAuthenticationRoles`** `(KeyM_CertificateIdType certId, KeyM_CertElementIdType certElementId, uint8 *role, uint32 *roleLength)`	
Parameter	
`certId`	Holds the identifier of the certificate.
`certElementId`	Specifies where the role information is located within the certificate.
`role`	The role data to be read by DCM.
`roleLength`	As IN parameter contains the maximum data length of the buffer for the role information. As OUT parameter shall return the actual length of the role. Its value should correspond to the setting in /Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationRoleSize
Return code	
Std_ReturnType	E_OK: Reading was successful, and data was provided in the buffer.
	E_NOT_OK: Reading was not successful.
Functional Description	
This API replaces the call of KeyM_CertElementGet() which would normally be called by DCM when an *Authentication (0x29)* sub-function 0x03 request is being processed. The purpose of this API is to allow the application to read the role information directly from the certificate by itself using the KeyM parameters certId and certElementId provided by this function. The role information shall then be transformed to the AUTOSAR format and passed back to DCM. This API is needed in the implementation of UDS service: > Authentication (0x29) when user specific authentication roles are enabled.	
Particularities and Limitations	
> ServiceID: 0xB6 > This function is synchronous. > This function is not reentrant.	

Table 5-82 Dcm_GetAuthenticationRoles()

5.5.3 Required Port Operation Functions

5.5.3.1 ConditionCheckRead()

Prototype	
`Std_ReturnType` **`ConditionCheckRead`** `( Dcm_OpStatusType OpStatus, Dcm_NegativeResponseCodeType* ErrorCode )`	
Parameter	
`OpStatus`	Status of the current operation.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application, if the conditions to read a data element are correct.	
Particularities and Limitations	
> ServiceID: 0x37 > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 5-83 ConditionCheckRead()

5.5.3.2 ReadDataLength()

Prototype	
`Std_ReturnType ReadDataLength ( Dcm_OpStatusType OpStatus, uint16 *DataLength )`	
Parameter	
`OpStatus`	Status of the current operation.
`DataLength`	Length of the data to be read.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. The DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to return the data length of a data element. Note: This callout type is available only if the DID has dynamic length.	
Particularities and Limitations	
> ServiceID: 0x36 > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 5-84 ReadDataLength()

5.5.3.3 ReadData() (asynchronous)

Prototype	
`Std_ReturnType ReadData ( Dcm_OpStatusType OpStatus, <Datatype> *Data )`	
Parameter	
`OpStatus`	Status of the current operation.
`Data`	Buffer where the read data shall be copied.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. The DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to get a data value of a DID/PID if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	
Particularities and Limitations	
> ServiceID: 0x3B > This function is not reentrant. > This function is asynchronous.	

Table 5-85 ReadData() (asynchronous)

5.5.3.4 ReadData() (synchronous)

Prototype	
`Std_ReturnType` **`ReadData`** `( <Datatype> *Data )`	
Parameter	
`Data`	Buffer where the read data shall be copied.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. E_NOT_OK: The operation has failed. The DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to get a data value of a DID/PID if DcmDspDataUsePort is set to USE_DATA_SYNCH_CLIENT_SERVER/ USE_DATA_SYNCH_FNC.	
Particularities and Limitations	
> ServiceID: 0x34 > This function is not reentrant. > This function is synchronous.	

Table 5-86 ReadData() (synchronous)

5.5.3.5 ReadData() (paged-data-reading)

Prototype	
`Std_ReturnType` **`ReadData`** `( Dcm_OpStatusType OpStatus, uint8 *Data, uint16 *DataLength )`	
Parameter	
`OpStatus`	Status of the current operation.
`Data`	Buffer where the read data shall be copied.
`DataLength`	As IN parameter contains the maximum number of bytes the application may write to the provided buffer. Note that this may exceed the total remaining length of the data. As OUT parameter shall return the actual data chunk length.
Return code	
`Std_ReturnType`	E_OK: The operation is finished; all data chunks are copied. DCM_E_PENDING: Current data chunk read operation is not yet finished. E_NOT_OK: The operation has failed. DCM_E_BUFFERTOOLOW: There was more data to be copied, but the provided buffer was not big enough to fit all of them. The `DataLength` parameter contains the amount of currently copied data.
Functional Description	
This function is a request from the DCM to the application to get a data value of a DID if DcmDspDataUsePort is set to USE_PAGED_DATA_ASYNCH_CLIENT_SERVER/ USE_PAGED_DATA_ASYNCH_FNC. For details on this API usage, please refer to chapter *8.24 How to Save RAM using Paged-Buffer for Large DIDs*.	
Particularities and Limitations	
> ServiceID: 0xA3 > This function is not reentrant. > This function is asynchronous.	

Table 5-87 ReadData() (paged-data-reading)

5.5.3.6 WriteData() (dynamic length)

Prototype	
`Std_ReturnType` **`WriteData`** `( const <Datatype> *Data, uint16 DataLength, Dcm_OpStatusType OpStatus, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`Data`	Buffer containing the data to be written.
`DataLength`	Length of the data to be written.
`OpStatus`	Status of the current operation.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to set the data value of a DID. Note: This callout type is available only if the DID has dynamic length.	
Particularities and Limitations	
> ServiceID: 0x3E > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 5-88 WriteData() (dynamic length)

5.5.3.7 WriteData() (static length)

Prototype	
`Std_ReturnType` **`WriteData`** `( const <Datatype> *Data, Dcm_OpStatusType OpStatus, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`Data`	Buffer containing the data to be written.
`OpStatus`	Status of the current operation.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to set the data value of a DID. Note: This callout type is available only if the DID has constant length.	
Particularities and Limitations	
> ServiceID: 0x35 > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 5-89 WriteData() (static length)

5.5.3.8 ReturnControlToECU()

Prototype	
`Std_ReturnType` **`ReturnControlToECU`** `( Dcm_OpStatusType OpStatus, <ControlMaskType> ControlMask, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	Status of the current operation.
`ControlMask`	Contains/points to the CEMR from request or equals 0xF..F (all signals) on lost session/security permissions.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: **This return value is not allowed to be used!** E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to return control of an IOControl back to the ECU. For details about the usage of the ControlMask parameter and the possible ControlMaskTypes, please refer to chapter *4.23.3 Implementation Aspects* of *InputOutputControlByIdentifier (0x2F).*	
Particularities and Limitations	
> ServiceID: 0x39 > This function is not reentrant. > This function is synchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC. > The DCM_E_PENDING return value is not allowed to be used since this operation shall always be executed synchronously. Refer to *4.23.3 Implementation Aspects* of *InputOutputControlByIdentifier (0x2F)* for details. > The “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 5-90 ReturnControlToECU()

5.5.3.9 ResetToDefault()

Prototype	
`Std_ReturnType` **`ResetToDefault`** `( Dcm_OpStatusType OpStatus, <ControlMaskType> ControlMask, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	Status of the current operation.
`ControlMask`	Contains/points to the CEMR from request.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to reset an IOControl to default value. For details about the usage of the ControlMask parameter and the possible ControlMaskTypes, please refer to chapter *4.23.3 Implementation Aspects* of *InputOutputControlByIdentifier (0x2F).*	
Particularities and Limitations	
> ServiceID: 0x3C > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC. > The “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 5-91 ResetToDefault()

5.5.3.10 FreezeCurrentState()

Prototype	
`Std_ReturnType` **`FreezeCurrentState`** `( Dcm_OpStatusType OpStatus, <ControlMaskType> ControlMask, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	Status of the current operation.
`ControlMask`	Contains/points to the CEMR from request.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to freeze the current state of an IOControl. For details about the usage of the ControlMask parameter and the possible ControlMaskTypes, please refer to chapter *4.23.3 Implementation Aspects* of *InputOutputControlByIdentifier (0x2F).*	
Particularities and Limitations	
> ServiceID: 0x3A > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC. > The “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 5-92 FreezeCurrentState()

5.5.3.11 ShortTermAdjustment()

Prototype	
`Std_ReturnType` **`ShortTermAdjustment`** `( const uint8 *ControlOptionRecord, Dcm_OpStatusType OpStatus, <ControlMaskType> ControlMask, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`ControlOptionRecord`	Control option parameter for the adjustment request.
`OpStatus`	Status of the current operation.
`ControlMask`	Contains/points to the CEMR from request.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to adjust the IO signal. For details about the usage of the ControlMask parameter and the possible ControlMaskTypes, please refer to chapter *4.23.3 Implementation Aspects* of *InputOutputControlByIdentifier (0x2F).*	
Particularities and Limitations	
> ServiceID: 0x3D > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC. > The “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 5-93 ShortTermAdjustment()

5.5.3.12 GetScalingInformation()

Prototype	
`Std_ReturnType` **`GetScalingInformation`** `( Dcm_OpStatusType OpStatus, uint8 *ScalingInfo, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	Status of the current operation.
`ScalingInfo`	Buffer where the read scaling info data shall be copied.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to read the scaling information of the corresponding data signal.	
Particularities and Limitations	
> ServiceID: 0x38 > This function is not reentrant. > This function is asynchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 5-94 GetScalingInformation()

5.5.3.13 Start()

Prototype	
`Std_ReturnType Start ( [<DataType> <ReqSignalName>,] Dcm_OpStatusType OpStatus, [<DataType> <ResSignalName>,] [uint16 (*)DataLength,] Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`<ReqSignalName>`	**Optional list of parameters.** Exists only if at least one request signal is defined in the configuration for this RID operation. For each signal there will be a dedicated function parameter of the data type derived from the configuration. The parameter name is the same as the ECUC data container name.
`OpStatus`	Status of the current operation.
`<ResSignalName>`	**Optional list of parameters.** Exists only if at least one response signal is defined in the configuration for this RID operation. For each signal there will be a dedicated function parameter of the data type derived from the configuration. The parameter name is the same as the ECUC data container name.
`DataLength`	**Optional parameter.** Exists only if either the last request or response signal has dynamic length. As IN parameter contains the current request length (for dynamic length RID requests). As OUT parameter shall return the actual response length (for dynamic length RID responses). The DCM will ignore the returned value on RIDs with static response length.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. DCM_E_FORCE_RCRRP: Forces a RCR-RP response. The service port will be called again once the RCR-RP response is sent. The OpStatus parameter will contain the transmission result. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to start a RID execution.	
Particularities and Limitations	
> ServiceID: N/A > This function is not reentrant. > This function is asynchronous.	

Table 5-95 Start()

5.5.3.14 Stop()

Prototype	
`Std_ReturnType Stop ( [<DataType> <ReqSignalName>,] Dcm_OpStatusType OpStatus, [<DataType> <ResSignalName>,] [uint16 (*)DataLength,] Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`<ReqSignalName>`	**Optional list of parameters.** Exists only if at least one request signal is defined in the configuration for this RID operation. For each signal there will be a dedicated function parameter of the data type derived from the configuration. The parameter name is the same as the ECUC data container name.
`OpStatus`	Status of the current operation.
`<ResSignalName>`	**Optional list of parameters.** Exists only if at least one response signal is defined in the configuration for this RID operation. For each signal there will be a dedicated function parameter of the data type derived from the configuration. The parameter name is the same as the ECUC data container name.
`DataLength`	**Optional parameter.** Exists only if either the last request or response signal has dynamic length. As IN parameter contains the current request length (for dynamic length RID requests). As OUT parameter shall return the actual response length (for dynamic length RID responses). The DCM will ignore the returned value on RIDs with static response length.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. DCM_E_FORCE_RCRRP: Forces a RCR-RP response. The service port will be called again once the RCR-RP response is sent. The OpStatus parameter will contain the transmission result. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to stop an already started RID execution. Note: The DCM will call this function even if the concrete RID was not started yet. The application shall take care about correct sequence execution.	
Particularities and Limitations	
> ServiceID: N/A > This function is not reentrant. > This function is asynchronous.	

Table 5-96 Stop()

5.5.3.15 RequestResults()

Prototype	
`Std_ReturnType RequestResults ( <DataType> <ReqSignalName>,] Dcm_OpStatusType OpStatus, [<DataType> <ResSignalName>,] [uint16 (*)DataLength,] Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`<ReqSignalName>`	**Optional list of parameters.** Exists only if at least one request signal is defined in the configuration for this RID operation. For each signal there will be a dedicated function parameter of the data type derived from the configuration. The parameter name is the same as the ECUC data container name.
`OpStatus`	Status of the current operation.
`<ResSignalName>`	**Optional list of parameters.** Exists only if at least one response signal is defined in the configuration for this RID operation. For each signal there will be a dedicated function parameter of the data type derived from the configuration. The parameter name is the same as the ECUC data container name.
`DataLength`	**Optional parameter.** Exists only if either the last request or response signal has dynamic length. As IN parameter contains the current request length (for dynamic length RID requests). As OUT parameter shall return the actual response length (for dynamic length RID responses). The DCM will ignore the returned value on RIDs with static response length.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. DCM_E_FORCE_RCRRP: Forces a RCR-RP response. The service port will be called again once the RCR-RP response is sent. The OpStatus parameter will contain the transmission result. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to read the routine result of a stopped RID execution.	
Particularities and Limitations	
> ServiceID: N/A > This function is not reentrant. > This function is asynchronous.	

Table 5-97 RequestResults()

5.5.3.16 GetSeed() (with SADR)

Prototype	
`Std_ReturnType GetSeed ( const uint8 *SecurityAccessDataRecord, Dcm_OpStatusType OpStatus, uint8 *Seed, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`Seed`	Points to the response seed data.
`SecurityAccessDataRecord`	Points to the request data. If the current security access level does not have any request data, the pointer is still valid (points behind the sub-function byte).
`OpStatus`	Status of the current operation.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to provide a security level specific seed.	
Particularities and Limitations	
> ServiceID: 0x44 > This function is not reentrant. > This function is asynchronous.	

Table 5-98 GetSeed() (with SADR)

5.5.3.17 GetSeed() (without SADR)

Prototype	
`Std_ReturnType` **`GetSeed`** `( Dcm_OpStatusType OpStatus, uint8 *Seed, Dcm_NegativeResponseCodeType* ErrorCode )`	
Parameter	
`Seed`	Points to the response seed data.
`OpStatus`	Status of the current operation.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to provide a security level specific seed.	
Particularities and Limitations	
> ServiceID: 0x45 > This function is not reentrant. > This function is asynchronous.	

Table 5-99 GetSeed() (without SADR)

5.5.3.18 CompareKey()

Prototype	
`Std_ReturnType` **`CompareKey`** `( const uint8 *Key, Dcm_OpStatusType OpStatus, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`Key`	Points to the requested key.
`OpStatus`	Status of the current operation.
`ErrorCode`	If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case E_NOT_OK is returned.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. DCM_E_COMPARE_KEY_FAILED: The received key is not a valid one. NRC 0x35/0x36 will be send accordingly. E_NOT_OK: The operation has failed. A concrete NRC shall be set.
Functional Description	
This function is a request from the DCM to the application to verify the requested security access level specific key. If the attempt counter feature is enabled in the configuration, there is a different in the behavior of attempt counter increment depending on the AUTOSAR version selected in the configuration. Since AUTOSAR version R19-11, the attempt counter is incremented only when CompareKey() returns DCM_E_COMPARE_KEY_FAILED. For previous versions, the attempt counter is incremented when CompareKey() returns either DCM_E_COMPARE_KEY_FAILED or E_NOT_OK. For details regarding the selection of AUTOSAR version, please refer to the chapter *8.13 How to Select DCM AUTOSAR Version*.	
Particularities and Limitations	
> ServiceID: 0x47 > This function is not reentrant. > This function is asynchronous.	

Table 5-100 CompareKey()

5.5.3.19 Indication()

5.5.3.19.1 DCM AUTOSAR Version Equal or Newer Than R19-11

Prototype	
`Std_ReturnType` **`Indication`** `( uint8 SID, const uint8 *RequestData, uint32 DataSize, uint8 ReqType, uint16 ConnectionId, Dcm_NegativeResponseCodeType *ErrorCode, Dcm_ProtocolType ProtocolType, uint16 TesterSourceAddress )`	
Parameter	
`SID`	The diagnostic service Id.
`RequestData`	Points to the request data. Points behind the service Id byte.
`DataSize`	The requested data length (without the SID byte).
`ReqType`	The type of the diagnostic request: 0 - physical request, 1 - functional request.
`ConnectionId`	Unique ID of the connection on which the request has been received.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
`ProtocolType`	The ID of the protocol on which the request was received.
`TesterSourceAddress`	The diagnostic client source address.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_REQUEST_NOT_ACCEPTED: The diagnostic service shall not be processed. No response will be sent. E_NOT_OK: The operation has failed. A concrete NRC shall be set, otherwise the DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to validate the received diagnostic service, additionally to the DCM internal validation.	
Particularities and Limitations	
> ServiceID: 0x65 > This function is not reentrant. > This function is synchronous.	

Table 5-101 Indication() (AR version >=R19-11)

5.5.3.19.2 DCM AUTOSAR Version Older Than R19-11

Prototype	
`Std_ReturnType` **`Indication`** `( uint8 SID, const uint8 *RequestData, uint16 DataSize, uint8 ReqType, uint16 SourceAddress, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`SID`	Refer to *Indication() (AR version >=R19-11).*
`RequestData`	Refer to *Indication() (AR version >=R19-11).*
`DataSize`	Refer to *Indication() (AR version >=R19-11).*
`ReqType`	Refer to *Indication() (AR version >=R19-11).*
`SourceAddress`	The diagnostic client source address.
`ErrorCode`	Refer to *Indication() (AR version >=R19-11).*
Return code	
`Std_ReturnType`	Refer to *Indication() (AR version >=R19-11).*
Functional Description	
Refer to *Indication() (AR version >=R19-11).*	
Particularities and Limitations	
> ServiceID: N/A > This function is not reentrant. > This function is synchronous.	

Table 5-102 Indication() (AR version < R19-11)

5.5.3.20 Confirmation()

5.5.3.20.1 DCM AUTOSAR Version Equal or Newer Than R19-11

Prototype	
`Std_ReturnType` **`Confirmation`** `( uint8 SID, uint8 ReqType, uint16 ConnectionId, Dcm_ConfirmationStatusType ConfirmationStatus, Dcm_ProtocolType ProtocolType, uint16 TesterSourceAddress )`	
Parameter	
`SID`	The diagnostic service Id.
`ReqType`	The type of the diagnostic request: 0 - physical request, 1 - functional request.
`ConnectionId`	Unique ID of the connection on which the response has been sent.
`ConfirmationStatus`	The type of the response message and the transmission status.
`ProtocolType`	The ID of the protocol on which the response was sent.
`TesterSourceAddress`	The diagnostic client source address.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. E_NOT_OK: The operation has failed. Has no effect on DCM.
Functional Description	
This function is a notification from the DCM to the application that a diagnostic service processing is finished.	
Particularities and Limitations	
> ServiceID: 0x66 > This function is not reentrant. > This function is synchronous.	

Table 5-103 Confirmation() (AR version >= R19-11)

5.5.3.20.2 DCM AUTOSAR Version Older Than R19-11

Prototype	
`Std_ReturnType` **`Confirmation`** `( uint8 SID, uint8 ReqType, uint16 SourceAddress, Dcm_ConfirmationStatusType ConfirmationStatus )`	
Parameter	
`SID`	Refer to *Confirmation() (AR version >= R19-11).*
`ReqType`	Refer to *Confirmation() (AR version >= R19-11).*
`SourceAddress`	The diagnostic client source address.
`ConfirmationStatus`	Refer to *Confirmation() (AR version >= R19-11).*
Return code	
`Std_ReturnType`	Refer to *Confirmation() (AR version >= R19-11).*
Functional Description	
Refer to *Confirmation() (AR version >= R19-11).*	
Particularities and Limitations	
> ServiceID: N/A > This function is not reentrant. > This function is synchronous.	

Table 5-104 Confirmation() (AR version < R19-11)

5.5.3.21 GetDTRValue()

Prototype	
`Std_ReturnType` **`GetDTRValue`** `( Dcm_OpStatusType OpStatus, uint16 *Testval, uint16 *MinLimit, uint16 *MaxLimit, DTRStatusType *Status )`	
Parameter	
`OpStatus`	Status of the current operation. Since the operation is synchronous, only possible value is DCM_INITIAL.
`Testval`	Returns the current test value.
`MinLimit`	Returns the minimum limit.
`MaxLimit`	Returns the maximum limit.
`Status`	Returns the TID status: > DCM_DTRSTATUS_VISIBLE: All returned values are valid. > DCM_DTRSTATUS_INVISIBLE: All returned values are invalid.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. E_NOT_OK: The operation has failed, DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to report the corresponding MID test data. If the data is currently not available, the Status parameter shall be set to INVISIBLE. DCM will send to the tester zero values.	
Particularities and Limitations	
> ServiceID: N/A > This function is not reentrant. > This function is synchronous.	

Table 5-105 GetDTRValue()

5.5.3.22 RequestControl()

Prototype	
`Std_ReturnType` **`RequestControl`** `( uint8 *OutBuffer, const uint8 *InBuffer)`	
Parameter	
`OutBuffer`	Points to the response routine control data. If the current TID does not have any data, the pointer is still valid (points behind the TID parameter).
`InBuffer`	Points to the request routine control data. If the current TID does not have any data, the pointer is still valid (points behind the TID parameter).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. E_NOT_OK: The operation has failed, DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to start a TID execution.	
Particularities and Limitations	
> ServiceID: N/A > This function is not reentrant. > This function is synchronous.	

Table 5-106 RequestControl()

5.5.3.23 GetInfotypeValueData()

Prototype	
`Std_ReturnType` **`GetInfotypeValueData`** `( Dcm_OpStatusType OpStatus, uint8 *DataValueBuffer [, uint8 *DataValueBufferSize] )`	
Parameter	
`OpStatus`	Status of the current operation.
`DataValueBuffer`	Points to the response of the VID data.
`DataValueBufferSize`	**Optional parameter.** Exists only in AR 4.2.2 or later enabled DCMs. The input value is the total/maximum size of the VID data (incl. NODI) in bytes, configured in DCMs ECUC file (refer to *4.8.4*). The output value is the current size of the VID data (incl. NODI) in bytes, returned by the application.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed, DCM sends NRC 0x22.
Functional Description	
This function is a request from the DCM to the application to read the corresponding vehicle information. As long as the data is temporarily not available, the DCM_E_PENDING code shall be returned. Once the data is available, the E_OK shall be used to acknowledge that. The returned data size (via `DataValueBufferSize`) shall always be less or equal to the value passed by DCM as input. Refer to chapter *8.31 How to Enable Support of OBD VIDs with Dynamic Length* for details.	
Particularities and Limitations	
> ServiceID: 0x60 (introduced first with AR 4.3.0 DCM SWS) > This function is not reentrant. > This function is asynchronous.	

Table 5-107 GetInfotypeValueData()

5.5.3.24 StartProtocol()

Prototype	
`Std_ReturnType` **`StartProtocol`** `( Dcm_ProtocolType ProtocolType [, uint16 TesterSourceAddress, uint16 ConnectionId] )`	
Parameter	
`ProtocolType`	The ID of the new protocol to be started.
`TesterSourceAddress`	**Optional parameter.** Exists only in AR R19-11 or later enabled DCMs. The diagnostic client source address.
`ConnectionId`	**Optional parameter.** Exists only in AR R19-11 or later enabled DCMs. Unique ID of the connection on which the request has been received.
Return code	
`Std_ReturnType`	E_OK: The protocol switch is allowed. DCM_E_PROTOCOL_NOT_ALLOWED: The old protocol shall not be stopped and the new one is not accepted, DCM sends NRC 0x22 to the new request. E_NOT_OK: Same as DCM_E_PROTOCOL_NOT_ALLOWED.
Functional Description	
This function is a request from the DCM to the application to get permission for switching to a new protocol. It is called each time a request from a diagnostic client belonging to a protocol other than the currently active one is received or for the very first diagnostic request (i.e. switches from no active protocol to any other supported one).	
Particularities and Limitations	
> ServiceID: 0x67 > This function is not reentrant. > This function is synchronous.	

Table 5-108 StartProtocol() (AR version >= R19-11)

5.5.3.25 IsDidAvailable()

Prototype	
`Std_ReturnType` **`IsDidAvailable`** `( uint16 DID, Dcm_OpStatusType OpStatus, Dcm_DidSupportedType *supported )`	
Parameter	
`DID`	The DID to be checked for active in the current range.
`OpStatus`	Status of the current operation.
`supported`	Returns the information whether the DID is a supported one: DCM_DID_SUPPORTED: Requested DID is a valid one; DCM_DID_NOT_SUPPORTED: Requested DID is not a valid one;
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. The DCM will treat the DID as unsupported one.
Functional Description	
This function is a request from the DCM to the application to get information whether the requested DID, from a supported DID range is really a valid one or not. Note: This operation is only available if the corresponding DID range has been specified to have gaps (i.e. not all DIDs within the range are valid ones).	
Particularities and Limitations	
> ServiceID: 0x3F > This function is not reentrant. > This function is asynchronous.	

Table 5-109 IsDidAvailable()

5.5.3.26 ReadDidData()

Prototype	
`Std_ReturnType` **`ReadDidData`** `( uint16 DID, uint8* Data, Dcm_OpStatusType OpStatus, uint16 *DataLength, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`DID`	The DID which data will be read.
`Data`	Buffer where the read data shall be copied.
`OpStatus`	Status of the current operation.
`DataLength`	Actual length of the read data.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. The DCM sends NRC 0x22, if ErrorCode is not set.
Functional Description	
This function is a request from the DCM to the application to get the data of a concrete DID within a DID range.	
Particularities and Limitations	
> ServiceID: 0x40 > This function is not reentrant. > This function is asynchronous.	

Table 5-110 ReadDidData()

5.5.3.27 WriteDidData()

Prototype	
`Std_ReturnType` **`WriteDidData`** `( uint16 DID, const uint8 *Data, Dcm_OpStatusType OpStatus, uint16 DataLength, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`DID`	The DID which data will be written.
`Data`	Buffer where the requested data shall be copied from.
`OpStatus`	Status of the current operation.
`DataLength`	Actual length of the data to be written.
`ErrorCode`	NRC to be sent in the negative response in case of failure (E_NOT_OK).
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. The DCM sends NRC 0x22, if ErrorCode is not set.
Functional Description	
This function is a request from the DCM to the application to write the requested data of a concrete DID within a DID range.	
Particularities and Limitations	
> ServiceID: 0x41 > This function is not reentrant. > This function is asynchronous.	

Table 5-111 WriteDidData()

5.5.3.28 GetSecurityAttemptCounter()

Prototype	
`Std_ReturnType` **`GetSecurityAttemptCounter`** `( Dcm_OpStatusType OpStatus, uint8 *AttemptCounter )`	
Parameter	
`OpStatus`	Status of the current operation.
`AttemptCounter`	Contains the stored attempt-counter value.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed. The counter value will be assumed to be zero. Note: The delay-timer could be started, depending on the configuration (see below).
Functional Description	
Once DCM is initialized, DCM requests this function per security level to get the stored attempt-counter value prior last power-down/reset event.	
Particularities and Limitations	
> ServiceID: 0x59 > This function is not reentrant. > This function is asynchronous. > Exists for a certain security level only if the security row „DcmDspSecurityAttemptCounterEnabled” specific parameter is enabled and the security level supports brute-force-attack prevention (i.e. delay counter/timer).	

Table 5-112 GetSecurityAttemptCounter()

5.5.3.29 SetSecurityAttemptCounter()

Prototype	
`Std_ReturnType` **`SetSecurityAttemptCounter`** `( Dcm_OpStatusType OpStatus, uint8 AttemptCounter )`	
Parameter	
`OpStatus`	Status of the current operation.
`AttemptCounter`	Contains the current attempt-counter value.
Return code	
`Std_ReturnType`	E_OK: The operation is finished. DCM_E_PENDING: The operation is not yet finished. E_NOT_OK: The operation has failed.
Functional Description	
Each time the corresponding security level counter value is changed, DCM will first notify the application calling this API to store the new value prior giving any result to the diagnostic client. Note: DCM cannot provide any failed-write counter behavior replacement. It is up to the application to provide at next *GetSecurityAttemptCounter()* call an appropriate counter value, resp. just E_NOT_OK. If this API fails to store the current counter value, the NRC sent back is still one of the appropriate ones 0x35 or 0x36.	
Particularities and Limitations	
> ServiceID: 0x5A > This function is not reentrant. > This function is asynchronous. > Exists for a certain security level only if the security row „DcmDspSecurityAttemptCounterEnabled” specific parameter is enabled and the security level supports brute-force-attack prevention (i.e. delay counter/timer).	

Table 5-113 SetSecurityAttemptCounter()

5.5.3.30 ProcessRequestDownload()

Prototype	
`Std_ReturnType` **`ProcessRequestDownload`** `( Dcm_OpStatusType OpStatus, uint8 DataFormatIdentifier, uint8 MemoryIdentifier, uint32 MemoryAddress, uint32 MemorySize, uint32 *BlockLength, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only.
`DataFormatIdentifier`	Compression and encrypting method
`MemoryIdentifier`	Identifier of the Memory Block (e.g. used if memory section distinguishing is needed). Note: If it is not used this parameter shall be set to 0.
`MemoryAddress`	Starting address of server memory where the data is to be written.
`MemorySize`	Number of bytes in the MemoryData.
`BlockLength`	Max number of bytes for one Dcm_ProcessTransferDataWrite().
`ErrorCode`	If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case E_NOT_OK is returned.
Return code	
`Std_ReturnType`	E_OK: Request was successful. E_NOT_OK: Request was not successful. DCM_E_PENDING: Request is not yet finished.
Functional Description	
This function is used to start a download process and is needed for the implementation of UDS service: > RequestDownload (0x34)	
Particularities and Limitations	
> ServiceID: 0x30 > This function is not reentrant. > This function is asynchronous.	

Table 5-114 Dcm_ProcessRequestDownload()

5.5.3.31 ProcessRequestUpload()

Prototype	
`Std_ReturnType ProcessRequestUpload ( Dcm_OpStatusType OpStatus, uint8 DataFormatIdentifier, uint8 MemoryIdentifier, uint32 MemoryAddress, uint32 MemorySize, uint32 *BlockLength, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only.
`DataFormatIdentifier`	Compression and encrypting method
`MemoryIdentifier`	Identifier of the Memory Block (e.g. used if memory section distinguishing is needed). Note: If it is not used this parameter shall be set to 0.
`MemoryAddress`	Starting address of server memory where the data is to be read.
`MemorySize`	Number of bytes to be read.
`BlockLength`	Max number of bytes for one Dcm_ProcessTransferDataRead().
`ErrorCode`	If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case E_NOT_OK is returned.
Return code	
`Std_ReturnType`	E_OK: Request was successful. E_NOT_OK: Request was not successful. DCM_E_PENDING: Request is not yet finished.
Functional Description	
This function is used to start an upload process and is needed for the implementation of UDS service: > *RequestUpload (0x35)*	
Particularities and Limitations	
> ServiceID: 0x31 > This function is not reentrant. > This function is asynchronous.	

Table 5-115 Dcm_ProcessRequestUpload()

5.5.3.32 ProcessTransferDataWrite()

Prototype	
`Dcm_ReturnWriteMemoryType` **`ProcessTransferDataWrite`** `( Dcm_OpStatusType OpStatus, uint8 MemoryIdentifier, uint32 MemoryAddress, uint32 MemorySize, const uint8 *MemoryData, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only. DCM_FORCE_RCRRP_OK: The enforced RCR-RP transmission has finished with success.
`MemoryIdentifier`	Identifier of the Memory Block (e.g. used if memory section distinguishing is needed). Note: If it is not used this parameter shall be set to 0.
`MemoryAddress`	Starting address of server memory where the data is to be written.
`MemorySize`	Number of bytes in the MemoryData.
`MemoryData`	Data to be written (Points to the diagnostic buffer in DCM).
`ErrorCode`	If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case DCM_WRITE_FAILED is returned.
Return code	
`Dcm_ReturnWriteMemoryType`	DCM_WRITE_OK: Write was successful. DCM_WRITE_FAILED: Write was not successful. DCM_WRITE_PENDING: Write is not yet finished. DCM_WRITE_FORCE_RCRRP: Enforce RCR-RP transmission (vendor extension).
Functional Description	
This function is used to write memory data identified by the parameter memoryAddress and memorySize. It is needed for the implementation of UDS service: > TransferData (0x36)	
Particularities and Limitations	
> ServiceID: 0x27 > This function is not reentrant. > This function is asynchronous.	

Table 5-116 Dcm_ProcessTransferDataWrite()

5.5.3.33 ProcessTransferDataRead()

Prototype	
`Dcm_ReturnReadMemoryType` **`ProcessTransferDataRead`** `( Dcm_OpStatusType OpStatus, uint8 MemoryIdentifier, uint32 MemoryAddress, uint32 MemorySize, uint8 *MemoryData, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only. DCM_FORCE_RCRRP_OK: The enforced RCR-RP transmission has finished with success.
`MemoryIdentifier`	Identifier of the Memory Block (e.g. used if memory section distinguishing is needed). Note: If it is not used this parameter shall be set to 0.
`MemoryAddress`	Starting address of server memory from which data is to be read.
`MemorySize`	Number of bytes to be read.
`MemoryData`	Read Data (Points to the diagnostic buffer in DCM).
`ErrorCode`	If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case DCM_READ_FAILED is returned.
Return code	
`Dcm_ReturnReadMemoryType`	DCM_READ_OK: Read was successful. DCM_READ_FAILED: Read was not successful. DCM_READ_PENDING: Read is not yet finished. DCM_READ_FORCE_RCRRP: Enforce RCR-RP transmission (vendor extension).
Functional Description	
This function is used to read memory data identified by the parameter memoryAddress and memorySize. It is needed for the implementation of UDS service: > TransferData (0x36)	
Particularities and Limitations	
> ServiceID: 0x26 > This function is not reentrant. > This function is asynchronous.	

Table 5-117 Dcm_ProcessTransferDataRead()

5.5.3.34 ProcessRequestTransferExit()

Prototype	
`Std_ReturnType` **`ProcessRequestTransferExit`** `( Dcm_OpStatusType OpStatus, const uint8 *transferRequestParameterRecord, uint32 transferRequestParameterRecordSize, uint8 *transferResponseParameterRecord, uint32 *transferResponseParameterRecordSize, Dcm_NegativeResponseCodeType *ErrorCode )`	
Parameter	
`OpStatus`	DCM_INITIAL: All In-parameters are valid. DCM_PENDING: All parameters are still valid. This is the subsequent function calls after DCM_E_PENDING has been returned. DCM_CANCEL: All In-parameters are still valid, but since this call is a final one it must be used to finalize any pending activities only.
`transferRequestParameterRecord`	Data to be read by the application (Points to the diagnostic buffer in DCM).
`transferRequestParameterRecordSize`	Length of transferRequestParameterRecord in bytes.
`transferResponseParameterRecord`	Data to be written to the DCM (Points to the diagnostic buffer in DCM).
`transferResponseParameterRecordSize`	When the function is called this parameter contains the maximum number of data bytes that can be written to the transferResponseParameterRecord. The function returns the actual number of written data bytes in transferResponseParameterRecord.
`ErrorCode`	If written by the application, a specific NRC will be sent back. This NRC is evaluated only in case E_NOT_OK is returned.
Return code	
`Std_ReturnType`	E_OK: Transfer was successful. E_NOT_OK: Transfer was not successful. DCM_E_PENDING: Transfer is not yet finished.
Functional Description	
This function is used to terminate a download or upload process and is needed for the implementation of UDS service: > RequestTransferExit (0x37) The function arguments are used according to [2].	
Particularities and Limitations	
> ServiceID: 0x32 > This function is not reentrant. > This function is asynchronous.	

Table 5-118 Dcm_ProcessRequestTransferExit()

5.6 Service Ports

5.6.1 Client-Server Interface

A client server interface is related to a Provide Port at the server side and a Require Port at client side.

5.6.1.1 Provide Ports on DCM Side

At the Provide Ports of the DCM the API functions described in *5.1.5* are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

The following sub-chapters present the Provide Ports defined for the DCM and the Operations defined for the Provide Ports, the API functions related to the Operations and the Port Defined Argument Values to be added by the RTE.

5.6.1.1.1 DCMServices

Operation	API Function	Arguments
GetActiveProtocol	*Dcm_GetActiveProtocol()*	OUT Dcm_ProtocolType ActiveProtocol, ERR{E_OK}
GetSesCtrlType	*Dcm_GetSesCtrlType()*	OUT Dcm_SesCtrlType SesCtrlType, ERR{E_OK}
GetSecurityLevel	*Dcm_GetSecurityLevel()*	OUT Dcm_SecLevelType SecLevel, ERR{E_OK}
ResetToDefaultSession	*Dcm_ResetToDefaultSession()*	ERR{E_OK}
GetSecurityLevelFixedBytes	*Dcm_GetSecurityLevelFixedBytes()*	IN Dcm_SecLevelType SecLevel, OUT uint8 FixedBytes, INOUT uint8 BufferSize ERR{E_NOT_OK, DCM_E_BUFFERTOOLOW}
SetActiveDiagnostic	*Dcm_SetActiveDiagnostic()*	IN boolean Active, ERR{E_OK}
GetRequestKind	*Dcm_GetRequestKind()*	IN uint16 TesterSourceAddress, OUT Dcm_RequestKindType RequestKind, ERR{E_NOT_OK}
VsgSetSingle	*Dcm_VsgSetSingle*	IN Dcm_VsgIdentifierType VsgId, IN Dcm_VsgStateType State, ERR{E_NOT_OK}
VsgSetMultiple	*Dcm_VsgSetMultiple*	IN const Dcm_VsgIdentifierType* VsgIdList, IN uint16 VsgListSize, IN Dcm_VsgStateType State, ERR{E_NOT_OK}

Operation	API Function	Arguments
VsgIsActive	*Dcm_VsgIsActive*	IN Dcm_VsgIdentifierType VsgId, OUT Dcm_VsgStateType* State, ERR{E_NOT_OK}
VsgIsActiveAnyOf	*Dcm_VsgIsActiveAnyOf*	IN const Dcm_VsgIdentifierType* VsgIdList, IN uint16 VsgListSize, OUT Dcm_VsgStateType* State, ERR{E_NOT_OK}
SetDeauthenticatedRole	*Dcm_SetDeauthenticatedRole*	IN uint16 connectionId, IN const Dcm_AuthenticationRoleType deauthenticatedRole, ERR{E_OK}

Table 5-119 DCMServices

5.6.1.2 Require Ports on DCM Side

At its Require Ports the DCM calls Operations. These Operations must be provided by the SWCs by means of Runnable Entities. These Runnable Entities implement the callback functions expected by the DCM.

The following sub-chapters present the Require Ports defined for the DCM, the Operations that are called from the DCM and the related Notifications, which are described in chapter *5.4.4*.

5.6.1.2.1 DataServices_<DataName>

Operation	Callout
ConditionCheckRead()	Rte_Call_DataServices_<DataName>_<Operation>
ReadData() (synchronous) / ReadData() (asynchronous) / ReadData() (paged-data-reading)	
ReadDataLength()	
WriteData() (static length) / WriteData() (dynamic length)	
ReturnControlToECU()	
ResetToDefault()	
FreezeCurrentState()	
ShortTermAdjustment()	
GetScalingInformation()	

Table 5-120 DataServices_<DataName>

5.6.1.2.2 RoutineServices_<RoutineName>

Operation	Callout
Start()	Rte_Call_RoutineServices_<RoutineName>_<Operation>
Stop()	
RequestResults()	

Table 5-121 RoutineServices_<RoutineName>

5.6.1.2.3 SecurityAccess_<SecurityLevelName>

Operation	Callout
GetSeed() (with SADR) / GetSeed() (without SADR)	Rte_Call_SecurityAccess_<SecurityLevelName>_<Operation>
CompareKey()	
GetSecurityAttemptCounter()	
SetSecurityAttemptCounter()	

Table 5-122 SecurityAccess_<SecurityLevelName>

5.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>

Operation	Callout
Indication()	Rte_Call_ServiceRequestManufacturerNotification_<SWC>_<Operation>
Confirmation()	

Table 5-123 ServiceRequestManufacturerNotification_<SWC>

5.6.1.2.5 ServiceRequestSupplierNotification_<SWC>

Operation	Callout
Indication()	Rte_Call_ServiceRequestSupplierNotification_<SWC>_<Operation>
Confirmation()	

Table 5-124 ServiceRequestSupplierNotification_<SWC>

5.6.1.2.6 DtrServices_<MIDName>_<TIDName>

Operation	Callout
GetDTRValue()	Rte_Call_DtrServices_<MIDName>_<TIDName>_<Operation>

Table 5-125 DtrServices_<MIDName>_<TIDName>

5.6.1.2.7 RequestControlServices_<TIDName>

Operation	Callout
RequestControl()	Rte_Call_RequestControlServices_<TIDName>_<Operation>

Table 5-126 RequestControlServices_<TIDName>

5.6.1.2.8 InfotypeServices_<VEHINFODATA>

Operation	Callout
GetInfotypeValueData()	Rte_Call_InfotypeServices_<VEHINFODATA>_<Operation>

Table 5-127 InfotypeServices_<VEHINFODATA>

5.6.1.2.9 CallbackDCMRequestServices_<SWC>

Operation	Callout
StartProtocol()	Rte_Call_CallbackDCMRequestServices_<SWC>_<Operation>

Table 5-128 CallbackDCMRequestServices_<SWC >

5.6.1.2.10 DataServices_DIDRange_<RangeName>

Operation	Callout
IsDidAvailable()	Rte_Call_DataServices_DIDRange_<RangeName>_<Operation>
ReadDidData()	
WriteDidData()	

Table 5-129 DataServices_DIDRange_<RangeName>

5.6.1.2.11 UploadDownloadServices

Operation	Callout
ProcessRequestDownload()	Rte_Call_UploadDownloadServices_<Operation>
ProcessRequestUpload()	
ProcessRequestTransferExit()	
ProcessTransferDataWrite()	
ProcessTransferDataRead()	

Table 5-130 UploadDownloadServices

5.6.2 Managed Mode Declaration Groups

DCM is a mode manager of the following modes.

ModeDeclarationGroup	Description
DcmDiagnosticSessionControl	Represents the diagnostic sessions from the DCM configuration.
DcmCommunicationControl_<ComM_CHANNEL_SNV>	For each ComM channel, there is a mode declaration group that represents the communication state of the channel.
DcmEcuReset	Represents the normal ECU reset modes.
DcmModeRapidPowerShutDown	Represents the extended ECU reset modes.
DcmControlDTCSetting	Represents the DTC setting state.
DcmSecurityAccess	Represents the security access level from the DCM configuration.
DcmAuthenticationState	Represents the authentication state for a specific connection.

Table 5-131 ModeDeclarationGroups managed by DCM

5.6.2.1 DcmDiagnosticSessionControl

Callout	Description
Rte_Switch_Dcm_DcmDiagnosticSessionControl_DcmDiagnosticSessionControl	Called each time a session change occurs. This call is only a notification and has no effect on any DCM diagnostic service processing. Invoked by *DiagnosticSessionControl (0x10)* or S3 timeout.

Table 5-132 DcmDiagnosticSessionControl callouts

Mode	Description
DEFAULT_SESSION	Represents the UDS Default session (initial state).
PROGRAMMING_SESSION	Represents the UDS Programming session.
EXTENDED_SESSION	Represents the UDS Extended session.

Mode	Description
</Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionRow> container's short-name	Any user defined session.

Table 5-133 DcmDiagnosticSessionControl modes

5.6.2.2 DcmCommunicationControl_<ComM_CHANNEL_SNV>

Callout	Description
Rte_Switch_Dcm_DcmCommunicationControl_<ComMChannelSNV>_ Dcm_DcmCommunicationControl_<ComMChannelSNV>	Called each time a communication state change on the corresponding channel occurs. This call is only a notification and has no effect on any DCM diagnostic service processing. Invoked by service *CommunicationControl (0x28)* or *DiagnosticSessionControl (0x10)* or S3 timeout.

Table 5-134 DcmCommunicationControl _<ComM_CHANNEL_SNV> callouts

Mode	Description
DCM_ENABLE_RX_TX_NORM	Reception and transmission of application messages is enabled (initial state).
DCM_ENABLE_RX_DISABLE_TX_NORM	Reception of application messages is enabled but their transmission is disabled.
DCM_DISABLE_RX_ENABLE_TX_NORM	Reception of application messages is disabled but their transmission is enabled.
DCM_DISABLE_RX_TX_NORMAL	Reception and transmission of application messages is disabled.
DCM_ENABLE_RX_TX_NM	Reception and transmission of network management messages is enabled.
DCM_ENABLE_RX_DISABLE_TX_NM	Reception of network management messages is enabled but their transmission is disabled.
DCM_DISABLE_RX_ENABLE_TX_NM	Reception of network management messages is disabled but their transmission is enabled.
DCM_DISABLE_RX_TX_NM	Reception and transmission of network management messages is disabled.
DCM_ENABLE_RX_TX_NORM_NM	Reception and transmission of application and network management messages is enabled.
DCM_ENABLE_RX_DISABLE_TX_NORM_NM	Reception of application and network management messages is enabled but their transmission is disabled.
DCM_DISABLE_RX_ENABLE_TX_NORM_NM	Reception of application and network management messages is disabled but their transmission is enabled.
DCM_DISABLE_RX_TX_NORM_NM	Reception and transmission of application and network management messages is disabled.

Table 5-135 DcmCommunicationControl_<ComM_CHANNEL_SNV> modes

5.6.2.3 DcmEcuReset

Callout	Description
Rte_Switch_Dcm_DcmEcuReset_DcmEcuReset	Called each time a power down state change occurs. This call is a notification but has effect on the DCM diagnostic service *EcuReset (0x11)* or *DiagnosticSessionControl (0x10)* processing. Invoked by *EcuReset (0x11)* or *DiagnosticSessionControl (0x10)* for bootloader related sessions.
Rte_SwitchAck_Dcm_DcmEcuReset_DcmEcuReset	Called after the Switch API is called to get the mode transition acknowledged prior continuing with the EXECUTE mode switch. Invoked by *EcuReset (0x11)* or *DiagnosticSessionControl (0x10)* for bootloader related sessions

Table 5-136 DcmEcuReset callouts

Mode	Description
NONE	No reset (initial state)
HARD	Hard reset target request (service 0x11 0x01)
KEYONOFF	KeyOnOff reset target request (service 0x11 0x02)
SOFT	Soft reset target request (service 0x11 0x03)
JUMPTOBOOTLOADER	Jump to bootloader reset target request (service 0x10 0x02 or any session with jump boot support)
JUMPTOSYSSUPPLIERBOOTLOADER	Jump to system supplier bootloader reset target request (service 0x10 0x02 or any session with jump boot support)
EXECUTE	Commits an already made reset target request.

Table 5-137 DcmEcuReset modes

5.6.2.4 DcmModeRapidPowerShutDown

Callout	Description
Rte_Switch_DcmModeRapidPowerShutDown_DcmModeRapidPowerShutDown	Called each time a power down state change occurs. This call is a notification but has effect on the DCM diagnostic service *EcuReset (0x11)* processing. Invoked by *EcuReset (0x11)*
Rte_SwitchAck_DcmModeRapidPowerShutDown_DcmModeRapidPowerShutDown	Called after the Switch API is called to get the mode transition acknowledged prior continuing with the EXECUTE mode switch. Invoked by *EcuReset (0x11)*

Table 5-138 DcmModeRapidPowerShutDown callouts

Mode	Description
ENABLE_RAPIDPOWERSHUTDOWN	Rapid shutdown is enabled (initial state) or Rapid shutdown is disabled (Service 0x11 0x04)
DISABLE_RAPIDPOWERSHUTDOWN	Rapid shutdown is disabled (Service 0x11 0x05)

Table 5-139 DcmModeRapidPowerShutDown modes

5.6.2.5 DcmControlDTCSetting

Callout	Description
Rte_Switch_Dcm_DcmControlDtcSetting_DcmControlDtcSetting	Called each time a DTC setting state change occurs. This call is only a notification and has no effect on any DCM diagnostic service processing. Invoked by *ControlDTCSetting (0x85)*, *DiagnosticSessionControl (0x10)* or S3 timeout.

Table 5-140 DcmControlDTCSetting callouts

Mode	Description
ENABLEDTCSETTING	DTC setting is enabled (initial state service 0x85 0x01 or *DiagnosticSessionControl (0x10)* or S3 timeout)
DISABLEDTCSETTING	DTC setting is disabled (service 0x85 0x02)

Table 5-141 DcmControlDTCSetting modes

5.6.2.6 DcmSecurityAccess

FAQ
This mode declaration group is vendor specific one and only available under certain circumstances. Please refer to chapter *8.18 How to Know When the Security Access Level Changes* for more details.

Callout	Description
Rte_Switch_Dcm_DcmSecurityAccess_DcmSecurityAccess	Called each time a security access level change occurs. This call is only a notification and has no effect on any DCM diagnostic service processing. Invoked by *SecurityAccess (0x27)* or *DiagnosticSessionControl (0x10)* or S3 timeout.

Table 5-142 DcmSecurityAccess callouts

Mode	Description
LockedLevel	Represents the UDS locked level (initial state).

Mode	Description
</Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow> container‘s short-name	Any user defined security access level.

Table 5-143 DcmSecurityAccess modes

5.6.2.7 DcmAuthenticationState

Callout	Description
Rte_Switch_Dcm_AuthenticationStateModeSwitchInterface_<DcmDslMainConnection>_DcmAuthenticationState	Called each time an authentication state change occurs. This call is only a notification and has no effect on any DCM diagnostic service processing. Invoked by *Authentication (0x29)*, S3 timeout or *Dcm_ResetToDefaultSession()*.

Table 5-144 DcmAuthenticationState callouts

Mode	Description
DCM_DEAUTHENTICATED	Represents the deauthenticated state (initial state).
DCM_AUTHENTICATED	Represents the authenticated state.

Table 5-145 DcmAuthenticationState modes

5.6.3 Sender-Receiver Interface

The following S/R interfaces are available in the current implementation:

5.6.3.1 DataServices_{DidName/DidDataName}

Type	Callout
DataServices (Read)	Rte_Read_DataServices_READ_<DidName/DidDataName>_data
DataServices (Write)	Rte_Write_DataServices_WRITE_<DidName/DidDataName>_data
DataServices (Read PR-Port)	Rte_Read_DataServices_<DidName/DidDataName>_data
DataServices (Write PR-Port)	Rte_Write_DataServices_<DidName/DidDataName>_data

Table 5-146 Sender-receiver interface of type DataServices

Supported data types are illustrated in *Table 8-23*.

5.6.3.2 IOControlRequest_{DidName/DidDataName}

Type	Callout
IOControlRequest (Read)	Rte_Read_IOControlRequest_READ_<DidName/DidDataName>_<DataElementName>
IOControlRequest (Write)	Rte_Write_IOControlRequest_<DidName/DidDataName>_<DataElementName>

IOControlRequest (Read PR-Port)	Rte_Read_IOControlRequest_<DidName/DidDataName>_<DataElementName>

Table 5-147 Sender-receiver interface of type IOControlRequest

<DataElementName> is either

- underControl. Refer to *Table 8-23* for supported data types,
- IOOperationRequest which is of a type struct. *Table 5-5* illustrates type definitons of inputOutputControlParameter associated with IOOperationRequest. Enabling control mask shall be handled as stated in *Table 5-7*, and
- controlState. Refer to *Table 8-23* for supported data types.

5.6.3.3 IOControlResponse

Type	Callout
IOControlResponse (Read)	Rte_Read_IOControlResponse_<DidName/DidDataName>_IOOperationResponse
IOControlResponse (IsUpdated)	Rte_IsUpdated_IOControlResponse_<DidName/DidDataName>_IOOperationResponse

Table 5-148 Sender-receiver interface of type IOControlResponse

Table *5-6* illustrates type definitions associated with IOOperationResponse.

6 Configuration

6.1 Configuration Variants

The DCM supports the configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-POST-BUILD-SELECTABLE
- VARIANT-POST-BUILD-LOADABLE
- VARIANT-POST-BUILD-LOADABLE-SELECTABLE

The configuration classes of the DCM parameters depend on the supported configuration variants. For their definitions please see the *Dcm_bswmd.arxml* file.

6.2 Configurable Attributes

The description of each configurable option is described within its online help in the DaVinci Configurator 5 tool.

7 Cybersecurity

This chapter describes relevant information for a secure integration and configuration, so-called Cybersecurity Manual Instructions (CMI), of this component to fulfil identified Technical Cybersecurity Requirements (TCR). Additionally, functional security dependencies to other components are described.

7.1 Configuration

7.1.1 CMI-Dcm-SessionTimerS3

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall use the following configuration option only if necessary and the implications and risks of using that feature are fully understood: /Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmSessionTimerS3Enabled

Rationale:

According to [11], when the server transitions from any non-default session to the default session then a locked security level shall be activated in the server. The $S3_{Server}$ timer ensures that the default session is automatically entered as soon as the tester, which owns the non-default session, is no longer present for a certain amount of time (5s). In case that the $S3_{Server}$ timer is de-activated, the server may stay for an indefinite amount of time in a non-default session and thus in an unlocked security level.

7.1.2 CMI-Dcm-SessionProtection

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall use the following configuration option only if necessary and the implications and risks of using that feature are fully understood: /Dcm/DcmConfigSet/DcmMiscellaneous/DcmAcceptingOtherClientsDuringNonDefaultSessionEnabled

Rationale:

According to [11], a security level can only be unlocked via service 0x27 during active non-default session. Due to the implemented session protection, it is ensured, that only the client owning the non-default session has access to the unlocked diagnostics elements. If this session protection is deactivated by the aforementioned configuration option, all clients of the same diagnostic protocol (DcmDslProtocolRow) share not only the active non-default session but also a potentially unlocked security level.

7.2 Runtime Interfaces: Application

7.2.1 CMI-Dcm-SecurityAccess

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall implement the callbacks required for the challenge-response mechanism of service 0x27 (SecurityAccess), and optionally callbacks required to store the number of false access attempts into non-volatile memory. Details can be found in chapter *4.17 SecurityAccess (0x27).*

Rationale:

According to [11], Diagnostic services for downloading/uploading routines or data into a server and reading specific memory locations from a server are situations where security access may be required. Improper routines or data downloaded into a server could potentially damage the electronics or other vehicle components or risk the vehicle's compliance to emission, safety, or security standards. With regard to the challenge-response procedure, the DCM relies on a sufficiently secure implementation of the aforementioned callbacks on the part of the application.

7.2.2 CMI-Dcm-SecurityBypass

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall ensure the correct usage of the API *Dcm_SetSecurityBypass()*, if configured.

Rationale:

Incorrect usage may result in an unintentional deactivation of all access restrictions to diagnostic services based on UDS service *SecurityAccess (0x27)*. The correct usage of the API is described in chapter *8.42.1 How to Support Bypass Mode for Security-Access.*

7.2.3 CMI-Dcm-GetRecoveryStates

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall ensure the correct usage of the API *Dcm_GetRecoveryStates()*, if configured.

Rationale:

Incorrect usage may result in an unintentional activation of an unlocked security level. The intended usage of the API is described in chapter *8.27 How to Recover DCM State Context on ECU Reset/Power On.*

7.2.4 CMI-Dcm-GetAuthenticationRoles

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall ensure the correct usage of the API *Dcm_GetAuthenticationRoles()*, if configured.

Rationale:

The API replaces the call of KeyM_CertElementGet(), which would normally be called by DCM to retrieve the authentication role information during processing of service Authentication (0x29) sub-function 0x03. Incorrect usage may result in an unintentional activation of authentication roles. The intended usage of the API is described in chapter *4.19 Authentication (0x29)* and *5.5.2.14 Dcm_GetAuthenticationRoles()*.

7.2.5 CMI-Dcm-SetDeauthenticatedRole

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall ensure the correct usage of the API

Dcm_SetDeauthenticatedRole(), if configured.

Rationale:

According to [5], it is in some use cases desirable, that the application set the role instead of using a diagnostic service 0x29 with its potentially time-consuming certificate parsing. The DCM provides this API to overwrite the configured deauthentication role. Incorrect usage may result in an unintentional activation of authentication roles. The intended usage of the API is described in chapter *4.19 Authentication (0x29).*

7.2.6 CMI-Dcm-AuthenticationBypass

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall ensure the correct usage of the API *Dcm_SetAuthenticationBypass(),* if configured.

Rationale:

Incorrect usage may result in an unintentional bypassing to the authentication role checking prior to accessing diagnostic entities based on UDS service *Authentication (0x29)*. The correct usage of the API is described in chapter *8.42.2 How to Support Bypass Mode for Authentication.*

7.2.7 CMI-Dcm-VariantPreconditionChecks

Technical Cybersecurity Requirements: TCR-MSRC-DiagAccessControl

Instruction:

The user of MICROSAR Classic shall be aware of the risks that arise from incorrect configuration of variant execution preconditions regarding session, security and authentication states of diagnostic entities.

Rationale:

The execution preconditions of diagnostic entities are meant to be invariant. Still, there are some special scenarios that must be considered. They are described in chapter *8.19.1.2.1 Handling of State Execution Preconditions of Variant Diagnostic Entities*.

7.2.8 CMI-Dcm-ActivateSecurityEventReporting

Technical Cybersecurity Requirements: TCR-MSRC-GenerateSecurityEvents

Instruction:

The user of MICROSAR Classic shall activate the security event reporting for the required security events in the configuration.

Rationale:

This triggers the component to report the events to the IdsM component. The configuration can include referencing IdsM events. More details can be found in chapter *8.45 How to report security events to IdsM.*

7.3 Runtime Interfaces: BSW

TCR	Depends on Component(s)	Comment
TCR-MSRC-DiagAccessControl	Csm	The crypto service module provides a wide range of cryptographic algorithms. The Csm is used for authentication calculation.
	KeyM	The key manager module provides support for certificate handling and APIs to realize authenticated diagnostics via certificates.
	NvM	A transition to authenticated state can only be done after the client successfully executed the authentication sequence. In some use cases as in production, a frequent power-on/power off sequence is performed. To keep the achieved authentication state over the power off, there is a dedicated mode rule requesting the Dcm to persist the authenticated state via NvM.
TCR-MSRC-GenerateSecurityEvents	IdsM	The Intrusion Detection System Manager provides a standardized interface for receiving notifications of on-board security events.

Table 7-1 Cybersecurity-relevant Dependencies for Runtime Interfaces

8 Using the DCM

This chapter shall give some examples and hints, how to handle common use cases of the DCM.

8.1 How to Reduce RAM Usage

All diagnostic services in DCM have a constant length so the DCM integrator person can perform a static buffer pre-calculation for finding the optimal buffer size.

Starting with DCM 7.01.00, DaVinci Configurator 5 provides a means to estimate each DCM buffer size, depending on the configured diagnostic services, accessible via this buffer. The buffer-to-service relation is determined by the DCM protocol configuration entity that refers to a certain diagnostic service table.

Caution
Depending on the DCM configuration the calculated buffer sizes are either **precise** or just **estimated** values. The calculation algorithm has the goal to assure the minimum value required so that each service, related to the validated buffer, can be requested and processed in its simplest form (e.g. on multiple DID reading, that at least one DID can be read). The worst case could also be calculated, but it will require too much RAM to be reserved unnecessarily (e.g. it is not considered to be possible to read N-times the largest DID with a single diagnostic request).

There are two calculation steps:

- The first one verifies that the validated buffer must be set at least to the proposed value (Error Message). Otherwise runtime errors may occur.
- The second one verifies whether with the currently set buffer size the DCM will be able to execute the diagnostic service or will ignore the request resp. send negative responses due to a lack of enough buffer space (Warning Message).

The buffer size calculation considers only diagnostic (sub-)services, that are internally handled by DCM. Once a diagnostic (sub-)service is redirected to an application handler, it will be excluded from the buffer size calculation. This is always the case regardless of the fact if the given diagnostic service was/is completely configured in the ECUC file (e.g. all related DIDs are available).

In *Table 8-1 Diagnostic services with non-trivial DCM Buffer size estimation calculation method* you can find information about the exactness of the buffer size calculation for each diagnostic services DCM can handle. From this table there are excluded all diagnostic services that have a trivial calculation formula (e.g. *DiagnosticSessionControl (0x10)*) or could only be implemented by the application (i.e. non-UDS user-defined diagnostic services or UDS services for which the DCM does not have any configuration details in order to make a meaningful estimation).

Diagnostic Service	Buffer Size Calculation Type	
	Request	Response
RequestCurrentPowertrainDiagnosticData (0x01)	Precise	Precise[1]
RequestPowertrainFreezeFrameData (0x02)	Precise	Precise[2]
RequestEmissionRelatedDTC (0x03)	Precise	Not estimated[3]
RequestOnBoardMonitorTestResults (0x06)	Precise	Precise[4]
RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle (0x07)	Precise	Not estimated[3]
RequestControlOfOnBoardSystemTestOrComponent (0x08)	Precise	Precise
RequestVehicleInformation (0x09)	Precise	Precise
RequestEmissionRelatedDTCsWithPermanentStatus (0x0A)	Precise	Not estimated[3]
ReadDTCInformation (0x19)	Precise	Precise[5] Not estimated[3]
ReadDataByIdentifier (0x22)	Precise[6]	Minimum estimation[7]
ReadMemoryByAddress (0x23)	Precise[8]	Minimum estimation[9]
ReadScalingDataByIdentifier (0x24)	Precise	Precise
SecurityAccess (0x27)	Precise	Precise
Authentication (0x29)	Minimum estimation[10]	Minimum estimation[10]
ReadDataByPeriodicIdentifier (0x2A)	Precise[6]	Precise[11]
DynamicallyDefineDataIdentifier (0x2C)	Minimum estimation[12]	Precise
WriteDataByIdentifier (0x2E)	Precise	Precise
InputOutputControlByIdentifier (0x2F)	Precise	Precise
RoutineControl (0x31)	Precise	Precise
RequestDownload (0x34)	Precise[13]	Minimum estimation[14]
RequestUpload (0x35)	Precise[13]	Minimum estimation[14]
TransferData (0x36)	Minimum estimation[9]	Precise
RequestTransferExit (0x37)	Not estimated	Not estimated
WriteMemoryByAddress (0x3D)	Minimum estimation[9]	Precise[8]
ControlDTCSetting (0x85)	Precise	Precise
ResponseOnEvent (0x86)	Precise	Precise

Table 8-1 Diagnostic services with non-trivial DCM Buffer size estimation calculation method

[1] Based on worst case: “response for six times the largest PID”.
[2] Only if all PIDs accessible via this service have configured PID data size (usually not set, since the DEM implements the PID data retrieval).
[3] Usually the paged-buffer response will be used, so the final response length is not that much relevant.

In some special cases like reading fault memory data the required buffer size cannot be estimated or a pessimistic prediction will be applied to guarantee that the ECU will always respond. Nevertheless, DCM provides a means to reduce the overall required buffer size for these scenarios with the paged-buffer feature, whose usage is described in the following two chapters.

Note
Even with the paged buffer it has to be ensured that the configured buffer size is at least as big as the biggest DID or Snapshot Data + protocol header (SID etc.) and at least as big as a TP frame.

In corner cases it might be necessary to add additional bytes to the estimated minimum to avoid deadlock situations.

8.1.1 Reading fault memory data

For reading fault memory data, the DCM offers the option to enable response paged buffer handling, that may reduce the overall required buffer by DCM. Enabling this option will lead to an increased code ROM usage in DCM due to the added functionality. The affected diagnostic services are:

- *ReadDTCInformation (0x19)*
- *RequestEmissionRelatedDTC (0x03)*
- *RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle (0x07)*
- *RequestEmissionRelatedDTCsWithPermanentStatus (0x0A)*

[4] Only if DCM knows the OBDMID configuration (refer to *8.30 How to Switch Between OBD DTR Support by DCM and DEM*). Otherwise only the worst case for “SupportedID” OBD MID will be considered.
[5] For all sub-functions with constant length (i.e. 0x01, 0x07, 0x09, 0x0B-0x0E, 0x11 and 0x12).
[6] It is considered that multiple-DIDs can be requested as per ECUC configuration. Refer to the service configuration chapter for details on the maximum number of DID that can be requested simultaneously in a single message.
[7] It is guaranteed that the largest configured not dynamically definable DID with no paged-buffer response can be read at least in a single DID request. Note: The (WWH-)OBD DIDs are not considered as in „[1]“.
[8] The configured ALFIDs are taken into account for this estimation. If no specifc ALFID(s) specified, the worst case (0x44 or 0x45 in case of MID usage) will be considered.
[9] It is guaranteed that at least one memory byte can be transfered.
[10] Depends on the presence of the optional max size parameters see *4.19.4 Configuration Aspects*.
[11] UUDT buffer size is not considered here. Only the USDT request/response messages.
[12] It is guaranteed that at least one source item (DID or memory block) can be requested for the corresponding definition function. Sub-function „clear“ is of course precisly calculated.
[13] Based on the maximum number of address and data bytes of all configured ALFIDs.
[14] Minimum one byte for the maximum data length.

Note
It is recommended always to keep the paged buffer option in DCM enabled to avoid situations where the tester would not be able to get a positive response when reading the fault memory content.

To enable the paged buffer handling in DCM for reading fault memory data, just set the configuration parameter to TRUE:

/Dcm/DcmConfigSet/DcmPageBufferCfg/DcmPagedBufferEnabled

8.1.2 Reading multiple DIDs in a single request

The situation here is different from Reading fault memory data case with standard AUTOSAR approach. In case the tester requests reading more data as the response buffer can handle, the DCM will respond with NRC 0x14 (ResponseTooLong) to avoid buffer overflow. The tester shall then use single-DID requests to get the data. Only one diagnostic service supports multiple DIDs in a single request, therefore just the following one is affected:

> *ReadDataByIdentifier (0x22)*

Using the MSR DCM, you have still an option that will allow you to save RAM also for multiple- or single-DID reading when the DIDs are too large even for a single-DID request. Please refer to *8.24 How to Save RAM using Paged-Buffer for Large DIDs* for details on this usage.

8.1.3 Exchanging data between OS tasks

In some circumstances data cannot be interchanged directly between DCM and a SW-C. That means, that the application cannot directly write into a buffer provided by DCM e.g. to supply requested data for a *ReadDataByIdentifier (0x22)* request. This is the case when DCM and SW-C are mapped to different OS tasks.

To realize an intertask communication the RTE provides a temporary buffer where the data can be written to. The size of the temporary buffer depends on the used SW-C port. After the data is written to the RTE temporary buffer, the entire buffer – regardless of the amount of data that was written into the buffer – is copied into the provided DCM internal buffer. Therefore, DCM needs to ensure that the provided buffer is large enough to hold the whole data – even if only a part of that data is required for the response message.

Since the DCM does not know whether any SW-C connected to is mapped to a different OS task, a multi OS task mapping is assumed by default. This setting can be overwritten, if it can be guaranteed that DCM and the connected SW-Cs are mapped to the same OS task:

/Dcm/DcmConfigSet/DcmDsp/DcmDspRteImplicitComEnabled

Caution
Setting this parameter to FALSE, although DCM and SW-C are mapped to different OS tasks, may lead to memory corruption!

Note
For each service 0x22 request, DCM must check whether the response data fits into the buffer. If not, DCM responds with NRC 0x14 (ResponseTooLong) to avoid buffer overflow. For this verification in case of Implicit RTE communication, DCM will always use the maximum length instead of the concrete length for DIDs with variable length. This circumstance must be considered during configuration of the required buffer size.

8.1.4 Restricting the access of DIDs and RIDs to specific protocols

The details can be found in *8.48 How to Use Protocol specific restriction for DIDs and RIDs.*

8.2 How to Reduce DCM Main-Function Run Time Usage

The DCM is designed and optimized for best possible response performance. This means the DCM main function will perform as much as possible operations per single activation to keep the P2 timings requirements. Additionally, the DCM internal code is optimized for short run time in order to lower the CPU burden during the many operations performed within a single DCM main function activation. But in cases where other BSWs are intensely involved in the service processing, such as the DEM during reading the fault memory information, the DCM can no more guarantee for total short run time execution. Therefore, the DCM offers a configuration option that may reduce the CPU peak load by limiting the number of iterations of an external BSW API.

After introducing the signal level access on DID data, the CPU peak load can be significantly affected also by services other than *ReadDTCInformation (0x19)*. Such services are:

- *ReadDataByIdentifier (0x22)*
- *ReadDataByPeriodicIdentifier (0x2A)*
- *DynamicallyDefineDataIdentifier (0x2C)*

Any of these allows multiple DIDs in a single request, that, depending on the total number of DIDs in a request and the corresponding number of signals in a single DID, can lead to really long execution times of the *Dcm_MainFunction()*.

To enable the run time limitation in DCM set up the configuration parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmMaxNumberIterationsPerTask

FAQ

There is no recommended default value for this parameter. It shall be measured during the integration by testing the worst case of the diagnostic services mentioned above.

Please note that a too low value of this parameter will lower the CPU usage to a minimum but will lead to long processing times of a diagnostic request. RCR-RP responses will be always sent, since the P2 times expire after a few *Dcm_MainFunction()* iterations. A compromise between performance and CPU usage can be found using the *Split Task Functions* concept. Using this approach, the worker task will be called more often than the timer task. This will help to achieve less CPU load per task activation and at the same time more work done per unit of a real time.

8.3 How to Force DCM to not Respond on Requests with Response SIDs

Generally, the DCM will replay to any physical addressed not supported service identifier with a negative response: NRC 0x11 (ServiceNotSupported). This includes also all service identifier from the diagnostic response Id range [0x40, 0x7F]∪[0xC0, 0xFF]. In some cases, it is not allowed to reply to any request service Id from this range.

To specify whether DCM shall reply to any diagnostic response Id or not, set up the configuration parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmRespondAllRequest.

8.4 How to Handle Multiple Diagnostic Clients Simultaneously

Normally, DCM is a single instance component. This means that once a diagnostic client has sent a request, the server (DCM) is busy until the processing of that request is finished. While busy, the DCM cannot handle in parallel other clients' requests. In such a situation the second client will not get a response.

If it is required to always send a response to a parallel client request, the DCM offers an option to send NRC 0x21 (BusyRepeatRequest) to any additional request to the main one.

To specify the DCM behavior on a multiple client environment, set up the configuration parameter:
/Dcm/DcmConfigSet/DcmDsl/DcmDslDiagResp/DcmDslDiagRespOnSecondDeclinedRequest

Since there will be reserved RAM for each client the DCM shall be able to communicate with, the DCM RAM usage may increase drastically for large number of configured DCM connections. Also, the DCM main function run time, needed to process all parallel connections, may increase significantly. In the practice, even if the DCM is configured to communicate with many clients, it is not necessary that all of them will send request to the same server at the same time. To optimize the RAM and run time resource usage of DCM, there is configuration option provided that limits the amount of in parallel handled diagnostic clients:
/Dcm/DcmConfigSet/DcmDsl/DcmDslDiagResp/DcmDslDiagRespMaxNumOfDeclinedRequests

Note
DCM can be configured to support parallel handling of requests from different clients. See chapter *8.12.2 Diagnostic Client(s) Processing Parallelization* for more details.

8.5 How to Restrict a Diagnostic Service Execution by a Condition

On a reception of a validly formatted diagnostic request DCM evaluates also with it associated diagnostic session and security access restrictions, defined in DaVinci Configurator 5.

In case of not matching required states, the DCM automatically rejects the request with the appropriate NRC.

Additionally, DCM can be configured to consider any ECU specific states, related to a concrete diagnostic execution. These states are the so-called modes that can either be managed by any BSW, including DCM or a SWC. You can simply define a condition made of such a mode and create a rule that will be later used by a diagnostic service, sub-service, DID, RID, etc. as a processing restriction.

An example of a use case using mode rules is service *DiagnosticSessionControl (0x10)*. If you need to restrict session activation by an ECU condition, you must model this condition in your SWC and make a reference between the diagnostic session sub-service you want to restrict and a mode rule that uses this mode in a logical expression.

To configure any processing conditions and rule, refer to the configuration container in DaVinci Configurator 5: /Dcm/DcmConfigSet/DcmProcessingConditions

Later, you can reference these rules from the corresponding diagnostic processing object as an additional restriction to the diagnostic session and security access conditions.

8.6 How to Get Notified on a Diagnostic Service Execution Start and End

Usually, the DCM validates the requested services without involving the application, only using the configuration parameters. In some cases, the DCM application may need to know about a diagnostic services execution start and when it is finished. Additionally, the application may need to restrict globally the processing of all or just some diagnostic services.

For all the use cases mentioned above, the DCM offers two kinds of application notification groups:

- Manufacturer diagnostic service notification
- System supplier diagnostic service notification

Each of them supports a list of one or more request indication and response confirmation notification function pairs that will be called on request reception resp. service processing finishing time.

The differences between these two kinds of notifications are described in within the corresponding API documentation:

- *ServiceRequestManufacturerNotification_<SWC>*

- *ServiceRequestSupplierNotification_<SWC>*

To set up a manufacturer diagnostic service notification, add a configuration container in the DaVinci Configurator 5:

/Dcm/DcmConfigSet/DcmDsl/DcmDslServiceRequestManufacturerNotification

To set up a system supplier diagnostic service notification, add a configuration container in the DaVinci Configurator 5:

/Dcm/DcmConfigSet/DcmDsl/DcmDslServiceRequestSupplierNotification

FAQ
If you have already specified long lists of notifications and want just temporary to disable the usage of a certain kind of notifications (e.g. disable all manufacturer notifications), you don't need to delete the lists. Just disable the usage of the notification kind by setting up the corresponding DCM configuration parameter:
/Dcm/DcmConfigSet/DcmGeneral/DcmRequestManufacturerNotificationEnabled
/Dcm/DcmConfigSet/DcmGeneral/ DcmRequestSupplierNotificationEnabled

8.7 How to Limit the Diagnostic Service Processing Time

In general, there is no limitation of a diagnostic service processing time. If the DCM application needs longer time before it can return the final request result i.e. waiting for a response from an external ECU or during heavy NvM usage from other components, the DCM monitors the diagnostic P2 times and keeps the diagnostic client notified about the final response delay. This behavior fully complies with the ISO UDS specification.

In some cases, usually required by the car manufacturer, the DCM shall not wait endlessly for the final operation result, but instead it will have a configured service processing deadline. If such time monitoring is required, the time limit shall be set high enough, to avoid abortion of a long service execution. In such a situation the DCM will decouple the application, take over the service processing and finalize it with a specific NRC (usually 0x10 (GeneralReject)). In that way the diagnostic client will be notified about this critical situation and it will be given the opportunity to send a reset command to the server to reinitialize the ECU, since obviously the software is no more in a reliable state.

To enable the application reaction deadline monitoring, set up the DCM configuration parameter:
/Dcm/DcmConfigSet/DcmDsl/DcmDslDiagResp/DcmDslDiagRespMaxNumRespPend

8.8 How to Handle Characteristics at Session Change with Switch to/from FBL and ECU Reset

8.8.1 Characteristics for Jump to/from FBL (Service 0x10)

8.8.1.1 How to Jump into the FBL from Service DiagnosticSessionControl (Service 0x10)

To transition into the FBL from the ECU's application software, a diagnostic session change with enabled switch to FBL has to be requested. The following parameter specifies which diagnostic session request will trigger this transition:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionRow/DcmDspSessionForBoot

8.8.1.2 How to Manage Response Handling on Jump to FBL

By default and in compliance with HIS, the final positive response for a diagnostic request with SID 0x10, which is configured to switch to FBL, will be sent by the FBL after the switch. For this the DCM stores all necessary information for the FBL (see also *5.5.2.5 Dcm_SetProgConditions()*) and resets the ECU without sending the final positive response to the request.

In some cases, and depending on the FBL used in the ECU, it may not be possible to send a final response from the FBL. In that case the DCM within the ECUs application software shall first send the final positive response to the diagnostic client and then jump into the FBL. To enable this behavior, the following parameter in the DCM configuration has to enabled:

/Dcm/DcmConfigSet/DcmGeneral/DcmResetToFblAfterSessionFinalResponseEnabled

8.8.1.3 How to Manage Response Handling on Jump from FBL

By default and in compliance with HIS, the DCM will, after returning from the FBL, check the provided information from the FBL (see also *5.5.2.6 Dcm_GetProgConditions()*) to determine if a final positive response is necessary. This behavior can be disabled and a final response from the DCM be suppressed by disabling the following parameter in the DCM configuration:

/Dcm/DcmConfigSet/DcmGeneral/DcmFinalResponseToFblEnabled

8.8.2 Characteristics for Service ECU Reset (Service 0x11)

By default, a positive response for Service *EcuReset (0x11)* is sent before the reset. This behavior can be adjusted for each sub-function separately by adapting the following parameter in the configuration, so that the positive response will be sent after the reset:

/Dcm/DcmConfigSet/DcmDsp/DcmDspEcuReset/DcmDspEcuResetRow/DcmResponseToEcuReset

8.8.3 DoIP Specific Characteristics for Jump from FBL and ECU Reset

By default, the DCM sends out the final positive response for the diagnostic request as soon as it is ready to do so. In case of DoIP it is necessary that the tester sets up the TCP connection and enables the routing before the DCM sends out the final positive response for the diagnostic request.

By enabling the following parameter in the DCM configuration, the DCM will additionally wait, until it is notified by the ComM module that the corresponding channel is ready (see also *5.4.2.4 Dcm_SetChannelReady()*), before sending the response:

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslChannelReadyIndicationEnabled

Note

> Please be aware that multiple connections per ComM channel are possible and the state will switch to “ready” as soon as one connection is established.

> Please be aware that the parameter exists for each connection of the corresponding ComM channel. For consistency make sure to configure all of these connections as desired.

8.9 How to avoid P2 Time Violation during ECU Reset Phase

In some cases, it is possible that P2 time violation occurs during the ECU reset phase when executing the following functions:

> The HIS compliant jump into FBL

> ECU reset when positive response is configured to be sent after reset

To avoid P2 time violation during the ECU reset phase, the DCM can be configured to send an RCR-RP response prior to resetting the ECU. For that purpose, the DCM configuration parameter shall be set accordingly:
/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmSendRespPendOnRestart

8.10 How to Put DCM in a Non-Default Session at ECU Power-On

The DCM supports also the HIS compliant transition from FBL into the application software, where the positive response is to be sent after the transition is accomplished. These usually are responses for diagnostic services that cause a reset in the FBL during the reprogramming process: 0x10 0x01 and 0x11 0x01.

This mechanism can be used to instruct DCM to enter in a non-default session, using appropriate combination of the parameter values returned by the *Dcm_GetProgConditions()*. The callout shall return the value DCM_WARM_START to notify DCM that the out-parameters are valid and shall be evaluated. The correct values during this operation are defined below:

Member of the Dcm_ProgConditionsType parameter	Value
ConnectionId	Unique id of the connection on which the request has been received. This parameter is needed only when generic connections are configured, the configuration is in post-build loadable phase, or the AUTOSAR version of the configuration is R19-11 or newer. The details regarding generic connection can be found at *How to Configure Generic Connections.* This parameter is assigned per main connection in DaVinci Configurator 5: /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolRxConnectionId

Member of the Dcm_ProgConditionsType parameter	Value
TesterSourceAddr	Source address of the received request if meta data is enabled, otherwise the value as configured in /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolRxTesterSourceAddr
ProtocolId	Not evaluated.
Sid	0x10
SubFuncId	The Id of the session to be activated [0x02 -0x7E]. Must be a supported session within the DCM configuration (refer to *4.10 DiagnosticSessionControl (0x10)* for details).
ReprogrammingRequest	FALSE (Not evaluated.)
ApplUpdated	FALSE
ResponseRequired	FALSE

Table 8-2 Initialization of the Dcm_ProgConditionsType for non-default session activation at ECU power-on

8.11 How to Support Calibratable Configuration Parameters

Vector DCM provides a limited functionality for configuration calibration. The following chapters describe which DCM objects are possible to be calibrated after ECU programming.

8.11.1 OBD Calibration

DCM implementation is prepared for post-programming calibration regarding the OBD supported services and their sub-service parameters. With these calibration abilities you can only disable or re-enable an already configured and supported OBD service and/or any of its sub-service parameters. The following calibration levels are supported in DCM:

- Deactivate/Re-activate an OBD diagnostic service or complete disabling of OBD support
- Deactivate/Re-activate specific OBD related parameter identifiers
  - For [12]: PIDs/MIDs/TIDs/VIDs
  - For OBD in UDS resp. [13] and [14]:
    - DIDs in range 0xF400-0xF8FF
    - RIDs in range 0xE000-0xE1FF

8.11.1.1 Calibration of Supported OBD Services

DCM supports this level of calibration only in connection with the *How to Get Notified on a Diagnostic Service Execution Start and End* feature. It is recommended to use the *ServiceRequestManufacturerNotification_<SWC>* notification to block as early as possible any not supported OBD service identifiers.

Caution
Do not block any UDS OBD services: 0x22 and 0x31. These services are shared between OBD and the UDS protocol. In case OBD or/and the UDS OBD parameters shall be disabled, please refer to the chapter *8.11.1.2 Calibration of Supported OBD Parameter Identifier* to disable only the affected sub-service parameters.

The diagnostic service level filtering is completely handled by the application implementation. This can be achieved by a calibratable filter object that will be evaluated within the diagnostic request indication function. This application call shall behave depending on the filter state as follows:

> Any OBD service(s) is (are) **disabled**: set the ErrorPtr function parameter to **NRC 0x11** (SNS) and return the value **DCM_E_NOT** to DCM. On functional requests there will be no response sent back.

> Any OBD service(s) shall be **re-enabled**: just return the value **E_OK** to DCM.

FAQ
Filtering the OBD services on SID level within the *ServiceRequestManufacturerNotification_<SWC>* will avoid the diagnostic session transition into the default session, required on an OBD request. This is especially useful when the OBD support shall be completely disabled, and the ECU shall behave as if it is a general UDS ECU.

8.11.1.2 Calibration of Supported OBD Parameter Identifier

Due to the OBD protocol specifics, the filtering of single OBD related parameter identifier is completely handled within the DCM. The application shall implement only the write operation onto the calibratable DCM configuration objects described in *Table 8-3 Calibratable OBD "availability parameter identifier" values*.

There are two types of OBD parameter identifiers:

> Availability Parameter Identifier (APID):

  > For [12]: 0x00, 0x20, 0x40, … 0xE0

  > For OBD in UDS resp. [13] and [14]: 0xZZ00, 0xZZ20, 0xZZ40, … 0xZZE0, where ZZ stays for:

    > DIDs: Any value in range 0xF4-0xF8

    > RIDs: Any value in range 0xE0-0xE1

> Data Parameter Identifier (DPID): Any other parameter identifiers

The first type reports to the requester a bit map of the corresponding "data parameter identifiers" supported by the ECU. These bitmap values must always be consistent with the real ECU "data parameter identifier" availability configuration. To guarantee this consistency and simplify the calibration process, DCM uses calibratable bitmaps for each "availability parameter identifier" that shall be supported.

The following table shows the overview of all OBD diagnostic service dependent calibrateble symbols:

Diagnostic Service ID	Table Name	Availability Condition
0x01	Dcm_CfgSvc01SupportedIdMask[n] 1)	If SID 0x01 is to be supported.
0x02	Dcm_CfgSvc02SupportedIdMask[8] 2)	If SID 0x02 is to be supported
0x06	Dcm_CfgSvc06SupportedIdMask[n] 1)	If SID 0x06 is to be supported.
0x08	Dcm_CfgSvc08SupportedIdMask[n] 1)	If SID 0x08 is to be supported.
0x09	Dcm_CfgSvc09SupportedIdMask[n] 1)	If SID 0x09 is to be supported.
0x22	Dcm_CfgSvc22SupportedIdMask[n] 3)	If SID 0x22 with any OBD DIDs is to be supported.
0x31	Dcm_CfgSvc31SupportedIdMask[n] 4)	If SID 0x31 with any OBD RIDs is to be supported.

Table 8-3 Calibratable OBD “availability parameter identifier” values

1) n = total number of APIDs for this service.

2) always contains all possible APIDs.

3) n = total number of APIDs for the whole range of OBD DIDs [0xF400-0xF8FF].

4) n = total number of APIDs for the whole range of OBD RIDs [0xE000-0xE1FF].

All the above table symbols have a 32bit value according to [12] that represents the bitmap for the corresponding parameter identifier range, defined by the APID. The only identifier not available in these bitmaps is the APID 0x00, since this one shall always be supported if the corresponding OBD diagnostic service is to be supported. For example, if SID 0x02 is to be supported, then PID 0x00 must exist in order SID 0x02 to be able to report the complete parameter identifier support list. Due to the differences between the two-byte UDS OBD DIDs/RIDs and their single byte OBD equivalence, the following shall be considered for their calibration:

> If an OBD parameter identifier is to be **disabled**, its corresponding APID bit value in the bitmap shall be reset. For the two types of parameter identifiers this means:

> For an APID:

> All bits in the corresponding service table shall be reset as follows:

All APIDs below the one to be disabled shall reset bit 0.

The APID to be disabled and the greater ones shall have zero mask value.

Example: for SID 0x**02** APID 0x40 shall be disabled:

```
Dcm_CfgSvc02SupportedIdMask [4] =
{
   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX0b  /* APID 0x00*/
   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX0b  /* APID 0x20*/
```

```
    0000 0000 0000 0000 0000 0000 0000 0000b  /* APID 0x40*/
    0000 0000 0000 0000 0000 0000 0000 0000b  /* APID 0x60*/
};
```

Note
Disabling APID 0x00 would mean that the corresponding OBD diagnostic service is not available. Therefore actually the SID level filtering described in *8.11.1.1 Calibration of Supported OBD Services* shall apply.

> For a DPID: The corresponding APID table entry (table index = DPID / 32) bitmap value shall be changed (reset bit number [DPID % 32]).

Example: If PID 0x**51** of SID 0x**02** shall be disabled, then the value shall be:

```
Dcm_CfgSvc02SupportedIdMask [4] =
{
  XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX1b  /* APID 0x00*/
  XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX1b  /* APID 0x20*/
  XXXX XXXX XXXX XXX0 XXXX XXXX XXXX XXX1b  /* APID 0x40*/
  XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX0b  /* APID 0x60*/
};
```

> If a UDS OBD parameter identifier is to be **disabled**, its corresponding APID bit value in the bitmap shall be reset. Here are the same rules as for the single byte OBD APIDs to apply, but only within a concrete OBD DID type (i.e. 0xF4**XX**, 0xF6**XX**, etc.).

Example: If PID 0x**F600** for SID 0x**22** shall be disabled, then the value shall be:

```
Dcm_CfgSvc22SupportedIdMask[x] =
{
  XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX1b  /* APID 0xF400*/
  XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX0b  /* APID 0xF420*/
  0000 0000 0000 0000 0000 0000 0000 0000b  /* APID 0xF600*/
  0000 0000 0000 0000 0000 0000 0000 0000b  /* APID 0xF620*/
  XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXX0b  /* APID 0xF800*/
};
```

Caution
DCM will react just as proper as the calibrated values are. This means that the generator of the calibration values is responsible for the correctness of the DCM configuration. Therefore, the following points must be considered during the new bitmap values' generation:

- The APID concatenation must be considered - see examples above how bit 0 of the corresponding APID masks changes.
- It is not possible to enable any APID or DPID that did not exist in the initial DCM configuration. If the newly generated calibration value sets a bit in a bitmap, which was not set in the initial configuration, DCM will report the calibrated APID value. But once the tester tries to read the DPID, corresponding to the wrongly set bit in the APID, DCM will react according to its initial configuration state – the DPID is not supported.
- If the OBD functionality shall be completely disabled, then:
  - The OBD services must be filtered as described in *8.11.1.1 Calibration of Supported OBD Services.*
  - The UDS OBD DIDs/RIDs shall be disabled by resetting **all** APID specific bitmap values.

Any faulty calibration will **not** cause any damage to the ECU or its software but will **lead to OBD diagnostic protocol violations**.

8.12 How and When to Configure Multiple Protocols

DCM provides means for supporting multiple diagnostic protocols in one configuration. There are several use cases, where multiple protocols shall be used in need of:

- *Diagnostic Client(s) Processing Prioritization*
- *Diagnostic Client(s) Processing Parallelization*
- *Client Specific Diagnostic Application Timings*
- *Diagnostic Service Firewall*

Please refer to the corresponding use case chapter below for details. Please note that all these use cases can also be combined.

8.12.1 Diagnostic Client(s) Processing Prioritization

If one or more diagnostic clients shall have privileged access over other clients (e.g. OBD2 client is more important than an OEM service tool), then all clients shall be grouped according to their priority. These groups are called in the DCM configuration "protocols" (/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow). Each protocol possesses a priority property (DcmDslProtocolPriority) that determines the group importance. Please refer to the online help of this setting for more details about it.

Once all clients that will communicate with the ECU were classified upon their importance, their connections

(/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection) must be assigned to the corresponding protocol.

FAQ
It is important to know, that in case of **protocol prioritization needed**, each protocol available in the DCM configuration shall refer to a **dedicated diagnostic buffer**. If two or more protocols do share the same buffer, no concurrent reception of diagnostic requests will be possible for clients assigned to these protocols. Only in case a non-default session is already started and the ECU is currently not processing any request, will give a client with higher priority the opportunity to get access over the ECU (please refer to *Table 8-6 Protocol prioritization during non-default session*).

Having specified the diagnostic protocols with their tester connections, corresponding buffers and priority, your ECU is ready to handle privileged requests.

Under the assumption that for all requests the activation of the new protocol is accepted (*StartProtocol()* returns E_OK), the handling of higher priority clients to lower priority ones (and vice versa) in DCM in different diagnostic sessions is shown in the matrixes below. The most important situations that can occur between two concurrent clients are focused by dedicated colors. Please, refer to *Table 8-4 Color legend to the protocol prioritization matrixes* for detailed explanation.

Color	Meaning
Blue	Focuses on the different behavior for a lower or equal priority client when the ECU is in the default or in a non-default session.
Green	Focuses on the situations where a lower or equal priority client will get an NRC 0x21.
Orange	Focuses on the situations where an active job of a client will be interrupted by a higher priority client.
Grey	A situation that can never occur due to reactions in the preceded cases.

Table 8-4 Color legend to the protocol prioritization matrixes

Hi-Prio Client (A) / Lo-Prio Client (B)	Idle	Rx Ongoing	Rx End	Service Processing	Tx Ongoing	Tx End (Post-Processing)
Idle		Receive request (A).	Start service processing (A).	Continue service processing (A).	Continue response transmission (A).	Do post-processing (A).
Rx Ongoing	Receive request (B).	Receive both requests.	Start service processing (A), continue reception (B).	Continue service processing (A), continue reception (B).	Continue response transmission (A), continue reception (B).	Do post-processing (A), continue reception (B).
Rx End	Start service processing (B).	Continue reception (A), start service processing (B).	Start service processing (A), send NRC 0x21[3] (B).	Continue service processing (A), send NRC 0x21[3] (B).	Continue response transmission (A), send NRC 0x21[3] (B).	Do post-processing (A), start service processing (B).
Service Processing	Continue service processing (B).	Continue reception (A), continue service processing (B).	Interrupt service processing [1] (B), do post processing [2] (B), start service processing (A).	N/A	N/A	N/A
Tx Ongoing	Continue response transmission (B).	Continue reception (A), continue response transmission (B).	Interrupt response transmission (B), do post processing [2] (B), start service processing (A).	N/A	N/A	N/A
Tx End (Post-Processing)	Do post-processing (B).	Do post-processing (B), continue reception (A).	Do post-processing (B), start service processing (A).	N/A	N/A	N/A

Table 8-5 Protocol prioritization during default session

Hi-Prio Client (A) / Lo-Prio Client (B)	Idle	Rx Ongoing	Rx End	Service Processing	Tx Ongoing	Tx End (Post-Processing)
Idle		Receive request (A).	Switch to default session. Start service processing (A).	N/A[5)]	N/A[5)]	N/A[5)]
Rx Ongoing	Receive request (B).	Receive both requests.	Switch to default session. Start service processing (A), continue reception (B).	N/A[5)]	N/A[5)]	N/A[5)]
Rx End	Start service processing (B).	Continue reception (A), start service processing (B).	Switch to default session. Start service processing (A), send NRC 0x21[3)] (B).	N/A[5)]	N/A[5)]	N/A[5)]
Service Processing	Continue service processing (B).	Continue reception (A), continue service processing (B).	Interrupt service processing [1)] (B), do post processing [2)] (B), switch to default session, start service processing (A).	N/A	N/A	N/A
Tx Ongoing	Continue response transmission (B).	Continue reception (A), continue response transmission (B).	Interrupt response transmission (B), do post processing [2)] (B), switch to default session, start service processing (A).	N/A	N/A	N/A
Tx End (Post-Processing)	Do post-processing (B).	Do post-processing (B), continue reception (A).	Do post-processing (B), switch to default session, start service processing (A).	N/A	N/A	N/A

Table 8-6 Protocol prioritization during non-default session if Lo-Prio Client (B) is session owner

Hi-Prio Client (A) / Lo-Prio Client (B)	Idle	Rx Ongoing	Rx End	Service Processing	Tx Ongoing	Tx End (Post-Processing)
Idle		Receive request (A).	Start service processing (A).	Continue service processing (A).	Continue response transmission (A).	Do post-processing (A).
Rx Ongoing	Receive request (B)[4].	Receive both requests.	Start service processing (A), continue reception (B)[4].	Continue service processing (A), continue reception (B)[4].	Continue response transmission (A), continue reception (B)[4].	Do post-processing (A), continue reception (B)[4].
Rx End	Send NRC 0x21[3] (B).	Continue reception (A), send NRC 0x21[3] (B).	Start service processing (A), send NRC 0x21[3] (B).	Continue service processing (A), send NRC 0x21[3] (B).	Continue response transmission (A), send NRC 0x21[3] (B).	Do post-processing (A), send NRC 0x21[3] (B).
Service Processing	N/A	N/A	N/A	N/A	N/A	N/A
Tx Ongoing	N/A	N/A	N/A	N/A	N/A	N/A
Tx End (Post-Processing)	N/A	N/A	N/A	N/A	N/A	N/A

Table 8-7 Protocol prioritization during non-default session if Hi-Prio Client (A) is session owner

1) If an operation is ongoing (i.e. any callout with an OpStatus parameter that already has been called with OpStatus == DCM_INITIAL), then this operation is called for a last time with OpStatus == DCM_CANCEL to stop any further job execution.
2) In case of interruption all configured confirmation functions (i.e. *ServiceRequestManufacturerNotification_<SWC>*, *ServiceRequestSupplierNotification_<SWC>*) will be called to finalize the jobs e.g. releasing semaphores, resources, etc. The confirmation status will be negative.
3) NRC 0x21 will be sent only if configured (refer to *8.4 How to Handle Multiple Diagnostic Clients Simultaneously*). Otherwise there will be no response at all.
4) The low priority request reception will be granted only if there shall be NRC 0x21 to be sent back (see [3]). Otherwise there will be no response at all.
5) Not applicable since Lo-Prio Client (B) is no longer session owner.

8.12.2 Diagnostic Client(s) Processing Parallelization

To reduce the interference between concurrent tester requests to a minimum, DCM offers the possibility to process multiple service requests simultaneously in the default session. Due to the required resources, this of course only works if the affected testers do not share the same protocol and buffer. According to [3], DCM shall internally serialize all asynchronous C/S interface or C function calls to the same port interface or C function during

parallel diagnostic services processing and return a pending to the reentrant caller. For example, it is unfavorable to allow read and write access simultaneously for the same data identifier. However, this is not always sufficient. Therefore, DCM parallelizes diagnostic services to different degrees. The handling of two clients with different priorities in default session is shown in the matrixes further below. The most important situations that can occur between two concurrent clients are focused by dedicated colors. Please, refer to *Table 8-8* for detailed explanation. During a non-default session, incoming requests are still prioritized based on the priority of the protocol as described in chapter *8.12.1 Diagnostic Client(s) Processing Prioritization.*

Color	Meaning
Blue	Focuses on the situations where an active job might be delayed due to internal thread synchronization. The client priority does not matter, "first come, first served" applies here.

Table 8-8 Color legend to the protocol parallelization matrix

Hi-Prio Client (A) / Lo-Prio Client (B)	Idle	Rx Ongoing	Rx End	Service Processing	Tx Ongoing	Tx End (Post-Processing)
Idle		Receive request (A).	Start service processing (A).	Continue service processing (A).	Continue response transmission (A).	Do post-processing (A).
Rx Ongoing	Receive request (B).	Receive both requests.	Start service processing (A), continue reception (B).	Continue service processing (A), continue reception (B).	Continue response transmission (A), continue reception (B).	Do post-processing (A), continue reception (B).
Rx End	Start service processing (B).	Continue reception (A), start service processing (B).	Start service processing of both requests.	Continue service processing (A), start service processing (B).	Continue response transmission (A), start service processing (B).	Do post-processing (A), start service processing (B).
Service Processing	Continue service processing (B).	Continue reception (A), continue service processing (B).	Start service processing (A), continue service processing (B).	Continue service processing of both requests.	Continue response transmission (A), continue service processing (B) [1].	Do post-processing (A), continue service processing (B) [1].
Tx Ongoing	Continue response transmission (B).	Continue reception (A), continue response transmission (B).	Start service processing (A), continue response transmission (B).	Continue service processing (A), continue response transmission (B) [1].	Continue response transmission of both requests[2].	Do post-processing (A), continue response transmission (B) [2].
Tx End (Post-Processing)	Do post-processing (B).	Continue reception (A), do post-processing (B).	Start service processing (A), do post-processing (B).	Continue service processing (A), do post-processing (B) [1].	Continue response transmission (A), do post-processing (B) [2].	Do post-processing of both requests [2].

Table 8-9 Protocol parallelization during default session

1) Some services lock their required resources not only for the duration of the service processing, but also during transmission and until post processing. However, the post processing itself is never delayed.

2) Both clients are either transmitting or post processing which means, that a serialization was not or is no longer necessary.

The following tables show a list of all supported services and their degree of parallelization in descending order.

Degree	Description
Full	The service runs entirely in parallel. Simultaneously requested diagnostic services do not delay the execution in terms of additionally required DCM main function cycles.
ID	All operations of a specific identifier (DID, RID, etc.) are locked as long as the identifier is used by a particular thread during service processing. UDS identifiers and their corresponding OBD counterparts are considered as a single identifier (e.g., DID 0xF801 and VID 0x01). For more details about the available operations see chapter *5.6.1.2 Require Ports on DCM Side*.
Memory	Only one service accessing memory by address can be executed at a time.
Service	Only one instance of the service can be executed at a time.
None	A diagnostic request to change in a non-default session can only be executed if no other service is currently in progress. DCM delays the session change request until all services currently ongoing are finished. However, as soon as a non-default session is pending, the parallelization mode is left, and the prioritization mode entered. If multiple clients requesting a non-default session simultaneously while in parallel mode, DCM rejects those ones with lower priority with NRC 0x22.
N/A	The service is not allowed in default session according to [10]. Hence, parallelization cannot take place.

Table 8-10 Degrees of parallelization

Service	Parallelization Degree
0x01	Full[15]
0x02	Full[15]
0x03	Full[16]
0x04	Full[16]
0x06	Full[15]
0x07	Full[16]
0x08	Full[15]
0x09	ID
0x0A	Full[16]
0x10	None
0x11	Service
0x14	Full[16]
0x19	Full[16]
0x22	Service, ID
0x23	Service, Memory
0x24	Service

Service	Parallelization Degree
0x27	N/A
0x28	N/A
0x29	Service
0x2A	N/A
0x2C	Service, ID, Memory
0x2E	Service, ID
0x2F	N/A
0x31	ID
0x34	N/A
0x35	N/A
0x36	N/A
0x37	N/A
0x3D	Memory
0x3E	Full[15]
0x85	N/A
0x86	Service

Table 8-11 Service specific parallelization degrees

[15] Service is synchronous and therefore no need for serialization.
[16] Parallel accesses to fault memory are synchronized by DEM. A dedicated DemClientId per protocol is required.

Caution
DCM does not synchronize services requested in parallel if they are implemented by the application. This gives the application full control over the degree of parallelization but must be considered during the design phase. As a MICROSAR extension, the `threadId` member of the `pMsgContext` parameter can be used to distinguish between calls of the same external service processor from different threads respectively protocols.

8.12.2.1 Configuration Aspects

> The parallel protocol processing feature can be enabled with the following parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmParallelProtocolProcessingEnabled

> In Post-build loadable configurations, this parameter should be considered: /Dcm/DcmConfigSet/DcmGeneral/DcmMaxNumberOfThreads

8.12.3 Client Specific Diagnostic Application Timings

If the ECU shall be able to communicate with clients that have the same importance, but some of the clients are connected to it via bus systems that cannot guarantee the default P2 timings, then these clients can be assigned to a dedicated protocol. The new protocol shall fulfill the following requirements:

> share the same diagnostic service table (same services are accessible)

> have the same priority to avoid any protocol preemption

> share the same buffer

Only the protocol specific P2 and P2Star specific parameters:

> /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmTimStrP2ServerAdjust

> /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmTimStrP2StarServerAdjust

shall be specified so that the RCR-RP messages can be sent in time to the corresponding clients.

8.12.4 Diagnostic Service Firewall

If the ECU shall allow only limited diagnostic service access to certain diagnostic clients, then the multi-protocol feature can be used to specify that.

FAQ
Diagnostic service firewalling support is limited to service identifier level. This means, that you can specify whether a service is visible to a client or not, but cannot hide specific sub-functions, DIDs, RIDs, etc. of a service. This also implies **that if a diagnostic service with a given SID is available in more than one diagnostic service table; all its corresponding properties must be identical in all instances of this service**. For example: it is not possible to specify different session and security access execution precondition for the same SID in different tables.

Each protocol refers to a specific diagnostic service table (/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslProtocolSIDTable) that contains all services visible to this protocol. So in case an OBD2 tester shall only be able to access the OBD2 services (SID 0x01-0x0A), and the service tester shall be able to access all UDS services and additionally *ClearEmissionRelatedDTC (0x04)*, then the DCM configuration shall look like as follows:

- There shall be two diagnostic service tables (/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable):
  - One for the UDS services and the SID 0x04;
  - One for all OBD2 services (incl. SID 0x04);
- There shall be two diagnostic protocols such as:
  - The “service tool” one:
    - shall refer to the UDS service table;
    - shall contain only the service tester connection
  - The OBD2 one:
    - shall refer to the OBD2 service table;
    - shall contain only the OBD2 tester connection

In such a configuration the UDS tester will always get NRC 0x11 (ServiceNotSupported) if any OBD2 request other than 0x04 is addressed physically. The OBD2 tester will never get access to the UDS services – will get either NRC 0x11 (peer-to-peer communication) or no response (on functionally addressed requests).

8.13 How to Select DCM AUTOSAR Version

DCM ensures the compatibility to different AUTOSAR versions. The following option is available for choosing the appropriate version in the configuration:

Dcm/DcmConfigSet/DcmGeneral/DcmAutosarVersion

The lowest selectable version is AUTOSAR version 4.2.1. In comparison to this version, newer AUTOSAR versions introduced several breaking changes regarding interfaces and port operations. The affected elements per selectable version are listed below:

Version Changed	Affected interfaces and port operations
Since 4.2.2	*<Module>_<DiagnosticService>()*
	<Module>_<DiagnosticService>_<SubService>()
	Dcm_ReadMemory()
	Dcm_WriteMemory()
	GetInfotypeValueData()
Since R19-11	*Confirmation()*
	Indication()
	StartProtocol()
	Dcm_SetProgConditions()
	CompareKey()

Table 8-12 Interfaces and port operations affected by changes related to AUTOSAR version 4.2.1

The following interfaces are not affected by that configuration option, but should be considered being deprecated since AUTOSAR version 4.2.2:

- *Dcm_ExternalProcessingDone()*
- *Dcm_ExternalSetNegResponse()*

Please see the corresponding chapters for more details.

8.14 How to Select DEM-DCM Interface Version

DCM supports DEM AR 4.2.1, AR 4.3.0 and AR 4.3.1 API. The API version selection is not performed automatically since there is not always a DEM available in the ECU configuration, but indeed there is one used in the software. Therefore, a vendor specific configuration parameter for selection of the DEM API version is introduced: /Dcm/DcmConfigSet/DcmGeneral/DcmDemApiVersion. For more details please refer to the online help of this parameter.

Caution
DEM AR 4.3.0 API is supported for downwards compatibility. The user defined memory selection is only available for DEM AR 4.3.1 API.

8.14.1 Setting the ClientId for DEM AR 4.3.0 and AR 4.3.1 API

When using the DEM AR 4.3.0 or AR 4.3.1 API, a *ClientId* must be specified for each protocol. This is done with the configuration parameter:

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDemClientRef

8.15 How to Support OBD and UDS over a Single Protocol

Usually if an ECU shall support OBD communication capabilities (i.e. OBD2 diagnostic protocol), it shall have a dedicated connection to an OBD tester. This allows protocol/diagnostic client prioritization (refer to *8.12 How and When to Configure Multiple*

Protocols) and guaranteed OBD task handling. Nevertheless, there are requirements on supporting both UDS and OBD over a shared diagnostic connection. In this case, no client prioritization can take place, but still the ECU shall reset any shortterm changes caused by an UDS tester right before. This task is automatically performed by DCM. Once a **functionally requested OBD service** is received (regardless of whether it is supported or not by the current ECU configuration), the ECU will enter the default session, just before the OBD request evaluation and execution starts. This automatic switch is only possible if all conditions below are fulfilled:

- There is at least one OBD service (i.e. SID in range [0x00-0x0F]) configured for DCM (as an internal or external service processor implementation).
- There is exactly one diagnostic protocol configured in DCM. If there are two or more connections, please use the multi-protocol prioritization mechanism with shared diagnostic buffers instead if possible (refer to *8.12 How and When to Configure Multiple Protocols*).

Note
Any received request with a SID in range [0x00-0x0F] will be treated as an OBD service and will force ECU to switch to the default session.

8.16 How to Use a User Configuration File

DCM has an advanced code configuration and code generation tool that completely sets up the module. However, in exceptional cases there is a need to complete or override some of the generated parameters. Most common such cases are workarounds for issues found after product's release.

Caution
User configuration file content must either be described in this manual or agreed by Vector prior using it in production code.

A user configuration file has no specific name. It can be any text file form e.g. Dcm.cfg. To use already created user configuration file within the DCM's code generation process, you must specify the full path to this file here:

/Dcm/DcmConfigSet/DcmGeneral/DcmUserConfigFile

8.17 How to Know When the Diagnostic Session Changes

There are situations where the ECU shall cancel all by the tester activated functions, when the diagnostic session changes. In some cases, DCM can handle this internally:

- *ReadDataByPeriodicIdentifier (0x2A)*
- *DynamicallyDefineDataIdentifier (0x2C)*
- *CommunicationControl (0x28)*
- *ControlDTCSetting (0x85)*

For other diagnostic services, such as

- *InputOutputControlByIdentifier (0x2F)* (will be automatically reset by DCM only on (re-)entering default session)
- *RoutineControl (0x31)*

this task must be performed by the application. For that purpose, DCM already notifies the application by invoking a mode switch for the mode declaration group *DcmDiagnosticSessionControl*.

Additionally, for better DCM integration flexibility, there is also another way an application located in a CDD can be notified – by a simple function call.

Whether the DCM shall notify about diagnostic session changes using simple function calls, you can specify by using configuration containers:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionCallback

For each callback you need, a dedicated container of the above type shall be configured for DCM. The parameter /Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionCallback/DcmDspSessionCallbackFnc will specify the function you want to be called by DCM. All these functions will have the prototype defined in chapter *5.5.1.1 <Diagnostic Session Change Notification Callback>*.

8.18 How to Know When the Security Access Level Changes

There are situations where the ECU shall cancel all by the tester activated functions, when they were secured and the security level changes. In some cases, DCM can handle this internally:

- *ReadDataByPeriodicIdentifier (0x2A)*
- *DynamicallyDefineDataIdentifier (0x2C)*

For other diagnostic services, such as

- *InputOutputControlByIdentifier (0x2F)* (will be automatically reset by DCM only on (re-)entering default session)
- *RoutineControl (0x31)*

this task must be performed by the application. For that purpose, the DCM can notify the application in several ways each time the security level performs a non-self-state-transition. An example for such a transition is “Level 1 → Locked”, but not “Locked → Locked”. The latter occurs when the default session has been re-activated.

The possible notifications are:

- *Invoking a Mode Switch*
- *Calling a Function Implemented Within a CDD Module*

Using these indications, the application may stop any running background routines that are secured.

FAQ

A security access level change can be triggered by any of the following events:

- Any diagnostic session change caused by:
  - Service *DiagnosticSessionControl (0x10)*
  - TesterPresent Timeout
  - Protocol Preemption
- A successfully processed security unlocking sequence with service *SecurityAccess (0x27)*

8.18.1 Invoking a Mode Switch

Whether the DCM shall notify about security access change using a mode switch, you can specify by configuration parameter:

/Dcm/DcmConfigSet/DcmGeneral/DcmSecurityLevelChangeNotificationEnabled

In case of state change, the DCM will invoke a mode switch for the mode declaration group *DcmSecurityAccess.*

8.18.2 Calling a Function Implemented Within a CDD Module

Whether the DCM shall notify about security access changes using simple function calls, you can specify by using configuration containers:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityCallback

For each callback you need, a dedicated container of the above type shall be configured for DCM. The parameter /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityCallback/DcmDspSecurityCallbackFnc will specify the function you want to be called by DCM. All these functions will have the prototype defined in chapter *5.5.1.2 <Security Access Change Notification Callback>.*

8.19 Post-build Support

The DCM is optionally capable of flexible configuration selection at run time. The following post-build variants are supported:

- variant switching at runtime - *Post-build selectable*
- variant calibration - *Post-build loadable*
- combination of both - *Post-build loadable selectable*

Note
Please refer to the basic software module description (*Dcm_bswmd.arxml*) file accompanying your delivery to find which parameters support post-build parametrization.

This information is also displayed in the DaVinci Configurator 5 tool.

8.19.1 Post-build Variance Level

For all the supported variants mentioned above, there is certain variance level that is covered by the module. Since the DCM can be logically divided into two main parts:

- *Communication Part*
- *Diagnostic Services Part*

we will define the level of variance for each of them separately.

8.19.1.1 Communication Part

DCM's communication part includes every parameter located under the configuration container with path /Dcm/DcmConfigSet/DcmDsl. The few non-post-build capable parameters are defined within the *Dcm_bswmd.arxml* file.

In general, the communication part of DCM handles configurations with:

- different amount of protocols or/and different protocol properties such as:
  - P2/P2 timing adjustments, priorities, buffer assignment, protocol ID, service table references, etc.
  - different number of connections or/and different connection parameters such as:
    - changed diagnostic message identifiers (e.g. multi-ECU use case using the same ECU for both left and right doors)
    - with or without periodic transmission (e.g. when periodic reading is not allowed within a variant (note: This will be possible once the diagnostic part of DCM becomes capable of variant switching)

What cannot be changed is the number of diagnostic buffers and their size. Since the size is used for the RTE ports (*ServiceRequestManufacturerNotification_<SWC>* and *ServiceRequestSupplierNotification_<SWC>*) it cannot change after compile time since RTE is not post-build capable.

8.19.1.2 Diagnostic Services Part

Note
If you have used the only PBS like option on diagnostic service level, provided in DCM 5.00.00 and later versions, as an alternative way to handle multiple diagnostic service variants (described in details in chapter "*8.29 How to Handle Multiple Diagnostic Service Variants*") you may now want to switch to the fully operational PBS support by DCM described here.

Since PBL variant handling on diagnostic service is not yet supported, OBD calibration is still the only way to change variants by calibrating data only operation.

DCM's diagnostic service part includes every parameter located under the configuration containers with path /Dcm/DcmConfigSet/DcmDsd and /Dcm/DcmConfigSet/DcmDsp.

In general, the PBS support in DCM is limited to the selection of the following diagnostic entities per ECU variant:

- Diagnostic Services
- Diagnostic Sub-Services
- DIDs (and their operations)
- RIDs
- Memory Ranges
- OBD PIDs
- OBD MIDs
- OBD TIDs
- OBD VIDs

So, you can only decide whether a certain diagnostic entity is available or not in a certain ECU variant. This implies that if an entity is available in more than one variant depending on its type it is not possible to:

- Vary its execution preconditions (i.e. Session and SecurityAccess state references)
- Specify different DID/RID etc. data layout and content
- Specify variant dependent periodic rates
- Specify variant dependent scheduler capacity
- Specify variant RoE events
- Specify RID specific sub-functions (i.e. disable only the Stop operation for a RID)
- Specify IO DID specific operation (i.e. disable only FreezeCurrentState for an IODID)
- Etc.

Although the *Dcm_bswmd.arxml* file already limits those ECUC configuration containers and parameters that are not meant to be variable, there are still some of them that for

specific reasons had to be defined as variant. Here is an abstract list of the parameter/container kinds that are specified as post-build related, because they can be absent in a variant, even if they shall not vary in their values:

Rules	Description
All the diagnostic entities, listed above as variant-capable (e.g. DIDs, RIDs, diagnostic services etc.) that have the same identifier in the variants they occur shall always have the same short name.	This is required to guarantee that the corresponding diagnostic entity properties that are not variable will remain constant in all variants. For example, all corresponding containers that represent a concrete DID must have the same short name in all variants the DID is available. In this way, the DID will have the same data layout in all variants.
Invariant Boolean parameters will be merged over all variants.	There are some Boolean parameters (e.g. DcmDspRoeInitOnDSC) that may be missing in a certain variant (e.g. RoE not supported) thus their multiplicity or the container they belong to is specified as post-build capable. The parameter itself but shall not change its value over all the variants it applies to. Depending on the parameter semantic, the final value for all variants will either be TRUE or FALSE or last is best.
Invariant Integer parameters will be calculated over all variants.	There are some Integer parameters (e.g. DcmDspMaxPeriodicDidToRead) that may be missing in a certain variant (e.g. where service *ReadDataByIdentifier (0x22)* is not supported) thus their multiplicity or the container they belong to is specified as post-build capable. The parameter value shall not change its value over all the variants it applies to. Depending on the parameter semantic, the final value for all variants will either be the minimum, maximum (DcmDspMaxPeriodicDidToRead) or last is best (DcmDspPowerDownTime).
All configuration entities with execution preconditions (i.e. Session/Security/ModeRules) that have the same identifier in the variants shall have the same precondition.	For example, a diagnostic service shall not vary its session state dependencies in different variants. For details on what shall be considered in case of execution precondition mismatches, please refer to chapter *0*.

OBD related UDS entities are always linked to their corresponding OBD entities when such are available.	MSR DCM always applies UDS-to-OBD automatic linking for those UDS DIDs and RIDs that have corresponding OBD PID/MID/TID or VID. In the context of multiple variants, there can be configurations that for example do have only OBD2 entities (e.g. PIDs) in one variant and OBD related UDS entities (e.g. DIDs) in another variant. In this case DCM will still link those matching UDS and OBD entities as it does in a single variant configuration. The advantage is – the application must implement only one data provider for the overlapping UDS and OBD entities.

Table 8-13 Post-build configuration rules on invariant DCM parameters

The only exception from this rule are the seed, key, and ADR length parameters for *4.17 SecurityAccess (0x27)*. They can be configured for each variant separately.

8.19.1.2.1 Handling of State Execution Preconditions of Variant Diagnostic Entities

The execution preconditions of diagnostic entities are meant to be invariant. Still, there are some special scenarios that must be considered.

Note
The configuration tool will detect inconsistencies regarding execution preconditions on related diagnostic entities and warn you with an appropriate message. The message IDs (DCM05010 - DCM05025) you may get and their explanations are listed in *9.2 Code Generation Time Messages*.

Only one kind of diagnostic entity will not be validated upon execution precondition mismatch: **memory ranges**.
If configured numerically DCM always calculates an optimized equivalent memory layout, based on the configured memory ranges and their access type (read/write) related preconditions. If there are overlapping memory areas with different preconditions, they will be merged into a corresponding single memory area with new preconditions that allow access to it under a certain state only if at least one variant resp. overlapping instance within the same variant allows the access in the given state.
If configured symbolically, DCM cannot determine the final memory layout during code generation, thus those memory ranges shall not overlap each other per DcmDspReadMemoryRangeByLabelInfo resp. DcmDspWriteMemoryRangeByLabelInfo.

A) A diagnostic entity has execution preconditions that refer to states not existing in some variants.

There exist different state groups (“Session”, “Security” and “Authentication”) for the execution preconditions, which manage the available states (session, security level or authentication roles) for a diagnostic entity (diagnostic service, DID, etc.). The existence of

the state groups depends on the existence of the corresponding diagnostic service (*DiagnosticSessionControl (0x10)*, *SecurityAccess (0x27)* and *Authentication (0x29)*) per variant.

Example 1: The state group Security is only available in some variants, since service *SecurityAccess (0x27)* is only available in these variants either.

There are two possible options to handle this scenario:

> A diagnostic entity is protected by an execution precondition for example “security level 1”. The corresponding state group (*SecurityAccess (0x27)*) is only available in Variant C. If this diagnostic entity is available in Variants A and B, the execution preconditions can never be met. Therefore, such a diagnostic entity will never be able to send a positive response and thus shall not even exist in the affected variant(s) (see Table 8-14).

Variants / Configuration	Variant A	Variant B	Variant C
Service 0x27			■
Diagnostic entity (e.g. service)			■
Execution precondition (security level) configured			Level 1
Invariant precondition merged over all variants	Level 1		

Table 8-14 Preconditions for diagnostic entities in one variant

> If the preconditions shall exist for all variants to enforce a consistent NRC behavior, the diagnostic entity may be configured over all variants but must refer to the identical preconditions in each variant (see Table 8-15)! As mentioned above, the diagnostic entity will never be able to send a positive response.

Variants / Configuration	Variant A	Variant B	Variant C
Service 0x27			■
Diagnostic entity (e.g. service)	■	■	■
Execution precondition (security level) configured	Level 1	Level 1	Level 1
Invariant precondition merged over all variants	Level 1		

Table 8-15 Preconditions for diagnostic entities in multiple variants

Example 2: The diagnostic entity is restricted by a specific session, security level or authentication role, which is not available in all variants, but the service (*DiagnosticSessionControl (0x10)*, *SecurityAccess (0x27)* and *Authentication (0x29)*) is available in all variants.

In this case, the diagnostic entity should not exist in the variant, where the restriction is not available. Otherwise, the Dcm will treat the empty precondition (in the variant, where the session/security level is missing) as “there is no precondition” and it will merge these states over all variants, allowing the diagnostic entity to be always (in all variants) accessible (Table 8-16).

Variants / Configuration	Variant A	Variant B	Variant C
Service 0x27	■		■
Diagnostic entity (e.g. service)	■		■
Execution precondition (security level) configured	No restriction		Level 1
Invariant precondition merged over all variants	No restriction		

Table 8-16 Merge of inconsistent preconditions

Caution

Dcm merges the execution preconditions for a diagnostic entity over all variants, if the corresponding state group (Session, Security or Authentication) exists in that variant. This results in a super-set of all execution preconditions of that diagnostic entity, if they are not invariant.

Empty preconditions in a variant are treated as “always allowed” and will lead to no restrictions over all variants.

B) The execution precondition depends on the preconditions of other related diagnostic entities.

To have a UDS compliant NRC prioritization, the execution preconditions on diagnostic service level are derived from their sub-service parameters (i.e., sub-function or parameter identifiers such as DIDs). In other words, a diagnostic service shall be allowed in a specific state if at least one of its sub-service parameters is allowed in this state.

Example:

For service *ReadDataByIdentifier (0x22)* it is true, that it shall be allowed in the default session if at least one of the readable DIDs shall be readable in the default session. Otherwise, the DID specific operation “read” will have a precondition that allows to be accessed, but any attempt to read the DID will fail, since it will be rejected on higher processing (diagnostic service) level.

Problem:

The problem the multi-variant handling faces is that if the only DID that has required service *ReadDataByIdentifier (0x22)* to be accessible in the default session does not exist in a new variant where other readable DIDs are still available, meaning service *ReadDataByIdentifier (0x22)* is still required to be available too. This automatically means that the diagnostic service shall lose its permission to be accessible in the default session within that variant. Due to the invariance of the diagnostic entity preconditions (i.e. once allowed and now not allowed), such configurations will cause warnings to be issued in the configuration tool.

Solution:

There is no real solution for such situations since the affected service cannot be removed from the variant. But …

… what would happen in such a configuration?

The ECU will still reject any unsupported in the given state diagnostic entity (in our example the concrete DID). The only difference will be the NRC sent back by the ECU when the variant without the readable in the default session DID is active (i.e. instead of expected NRC 0x7F, 0x31 will be sent). The advantage is that the ECU will have a constant behavior independent of the active variant and will send the same NRC as in the variant with the DID readable in the default session.

8.19.2 Initialization

All post-build variants have in common that DCM must be first correctly initialized with the concrete variant. Thus, the variant switching is only possible at run time during the module initialization phase. For that purpose the DCM API *Dcm_Init()* must be called with the appropriate configuration root pointer. Please refer to the API description for more details.

The configuration pointer is passed by the MICROSAR Classic EcuM based on the post-build configuration. If no MICROSAR Classic EcuM is used, the procedure of how to find the proper initialization pointers is out of scope of this document.

8.19.2.1 Error Detection and Handling

The DCM will verify the configuration data before accepting it to initialize the module. If this verification fails, an EcuM error hook (*EcuM_BswErrorHook*) is called with an error code according to *Table 8-17.*

Error Code	Reason
ECUM_BSWERROR_NULLPTR	Initialization with a null pointer.
ECUM_BSWERROR_MAGICNUMBER	Magic pattern check failed. This pattern is appended at the end of the initialization root structure. An error here is a strong indication of random data, or a major incompatibility between the code and the configuration data.

Error Code	Reason
ECUM_BSWERROR_COMPATIBILITYVERSION	The configuration data was created by an incompatible generator. This is also tested by verification of a ‘magic’ pattern, so initialization with random data can also cause this error code.

Table 8-17 Error Codes possible during Post-Build initialization failure

If no MICROSAR Classic EcuM is used, this error hooks and the error code constants must be provided by the environment. The DCM performs the following verification steps:

1. If the pointer equals NULL_PTR, initialization is rejected.
2. If the initialization structure does not end with the correct magic number, it is rejected.
3. If the initialization structure was created by an incompatible generator version, it is rejected (starting magic number check)

Caution
The verification steps performed during initialization are neither intended nor sufficient to detect corrupted configuration data. They are intended only to detect initialization with a random pointer, and to reject data created by an incompatible generator version.

8.19.3 Post-build Variants

8.19.3.1 Post-build selectable

The MICROSAR Classic Identity Manager (refer to [18]) is an implementation of the AUTOSAR 4 post-build selectable concept. It allows the ECU manufacturer to include several DCM configurations within one ECU. With post-build selectable and the Identity Manager the ECU variants are downloaded within the ECUs non-volatile memory (e.g. flash) at ECU build time. Post-build selectable does not allow modification of DCM aspects after ECU build time. At the same time, this limitation allows some of the optimization strategies still to be effective – DCM static code part will be optimized for the variant with maximum configuration size.

The variant selection is performed at run time by passing the corresponding configuration root during the module initialization (refer to chapter *8.19.2 Initialization*).

8.19.3.2 Post-build loadable

All DCM configuration parameters, that are classified to be post-build selectable, also do support post-build loadable variant. The differences to the post-build selectable case are listed upon their qualification:

> advantages:

  > The module’s configuration can be updated after the module’s compile time without reprogramming the whole ECU software.

> disadvantages:

- Since all the affected configuration parameters may change after module's compile time, the optimization level of the source code is very low.
- Since no maximum configuration size can be pre-calculated, some scalable RAM blocks are referred not by a direct linker symbol, but through a pointer.
- Only one configuration variant is supported at a time (no variant selection at run time possible). This disadvantage is avoided if the post-build loadable selectable variant is chosen instead (refer to chapter *8.19.3.3*).
- Greater risks of passing an invalid pointer during module initialization time.

For details about the post-build loadable feature, please refer to [17].

8.19.3.3 Post-build loadable selectable

This variant combines both post-build selectable and loadable variants, allowing a variant selection at run time and at the same time post-build calibration of parameters.

For details on the two mentioned variants, please refer correspondingly to chapters *8.19.3.1* and *8.19.3.2*.

8.19.3.4 Post-build deleteable

This variant is a specific sub-variant of the post-build loadable variants. It allows deleting of containers that were created at link time, by guaranteeing at the same time the preservation of other post-build capable parameters' values. For details about this feature, refer to the [17].

8.20 Handling with DID Ranges

8.20.1 Introduction

The DIDs in DCM are usually configured in detail: For a concrete DID number, it is specified the data access type (read, write, etc.), number of data signals the DID contains, etc. For each data signal it is exactly configured the maximum/concrete length and type of data acquisition (i.e. RTE C/S port, function call, direct NvM interaction, etc.).

Additionally, DCM supports a more generic DID access method, using DID ranges. This method has its advantages and disadvantages:

Advantages:

- You can implement only one service port/function that covers a large group of DIDs with a similar data access method.

Disadvantages:

- Only read and write operations are allowed when using DID ranges. No IO-control or scaling information reading is possible.

8.20.2 Implementation Limitations

Current AR DCM SWS ([1]) defines DID range interaction with the application in such a way that some restrictions must be considered when configuring a DID range.

- DID ranges may not be defined for DIDs 0xF300-0xF3FF (dynamically defined DIDs).
- DID ranges may not be defined for DIDs 0xF400-0xF8FF (OBD/ WWH-OBD DIDs), when DCM shall handle these on its own.

- If a DID from a DID range shall be included in a dynamically defined DID, the requested *DynamicallyDefineDataIdentifier (0x2C)* service will validate the source position and size parameters only upon the configured DID range maximum possible length (*8.20.3 Configuration Aspects*). Hence, when the actual length of the DID from this range is smaller than the maximum length and the stored source position and size do not match the actual length, the reported data will be fully or partially invalid.
- If a DID from a DID range is used in a multi DID request for service *ReadDataByIdentifier (0x22)*, in order to protect the ECU from out of boundary access during reading each, DCM will consider at first its maximum length for the total response length. Later, the application will return the concrete length during reading DID-Range data, so the positive response will always have the correct length. The only negative effect is that DCM may reject requests with multiple DIDs that would actually fit the configured buffer. So, choosing values for the maximum DID range length, nearly equal the size of the diagnostic buffer will mostly fail a multi DID request with a DID range DID. To avoid such situations, please consider the following guideline:
  - Use DID ranges for DIDs that have nearly the same size, which is represented by the maximum length parameter.
  - If not possible to group the DID in the way shown above, try splitting large ranges into smaller ones to have less differences between the shortest and longest DID of a range.
  - Try grouping short DIDs within ranges. If the maximum length of a DID range is far smaller than the diagnostic buffer, then the multiple DID request limitation will no longer persist. The best proportion is:
  - DCMBufferSize >= DcmDspMaxDidToRead * DcmDspDidRangeMaxDataLength
- The DID range response length calculation limits also the usage of the paged DIDs (*8.24 How to Save RAM using Paged-Buffer for Large DIDs*).
- Since DID ranges support read operation, they may be used for periodic reading, but then the maximum length may not exceed 7 bytes (CAN UUDT reference length).

8.20.3 Configuration Aspects

> If a DID ranges is readable or/and writeable the corresponding UDS services shall be defined in the configuration tool. Refer to *ReadDataByIdentifier (0x22)* and *WriteDataByIdentifier (0x2E)* for more information about their configuration aspects.

> Whether a DID range has read or/and write operation, is to be determined via a corresponding /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo container (referenced by /Dcm/DcmConfigSet/DcmDsp/DcmDspDidRange/DcmDspDidRangeInfoRef

Refer to the concrete DID range configuration parameter online help in the configuration tool for more details about the effect of the parameter value, dependencies to other configuration parameters or any specific restrictions.

8.21 How to Support DID 0xF186

The ActiveDiagnosticSessionDataIdentifier (0xF186) is used to report the active diagnostic session within the DCM. If you want DCM to implement the read access to its data, please follow the configuration steps below:

> A DID shall be defined within the following container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDid

> Set the identifier of that DID to 0xF186:
/DcmConfigSet/DcmDsp/DcmDspDid/DcmDspDidIdentifier

> Define a read operation for that DID:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidRead

> The read function should have the name “Dcm_DidMgr_F186_ReadData”:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataReadFnc

> Select the value USE_DATA_SYNCH_FNC for the following container:
/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort

> Because only one data byte must be read the data size should be configured to 8 bit:
/DcmDsp/DcmDspData/DcmDspDataSize

Note
Since this is just a regular DID, that can be used in arbitrary manner, the following must be considered if other options related to this DID are set:

> The DID may have also other data signals. If one of them fulfills to the above conditions, you can still use the DCM’s internal implementation for reporting current session ID.

> If the DID shall support any other operation than only read (e.g. write), then for the data signal, that will use the DCM’s internal implementation, the write operation must be implemented by the application.

  > An example for a write functionality: Since DCM does not provide an API for entering a non-Default session, the only effect such a write function may have is to put DCM into the default session (refer to *Dcm_ResetToDefaultSession()*) when the requested value is 0x01. All other values shall be rejected by NRC 0x31.

8.22 How to Suppress Responses to Functional Addressed Requests

Sometimes it may be necessary on a specific connection to suppress all kind of responses (positive or negative) on functional addressed service requests. This feature will be automatically activated when Mixed11 addressing (applies to CanTP only) is configured for that connection. To achieve this, the following addressing type parameter must be configured to “DCM_NET_ADDR_MIXED_11”:

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslAddressingType

Additionally, the following configuration switch must be enabled:

/Dcm/DcmConfigSet/DcmGeneral/DcmSuppressResponseOnCanTpFuncMixedAddrRequests

8.23 How to Support Interruption on Requests with Foreign N_TA

The DCM supports service processing interruption when a request from the same client to another ECU is detected. This feature is only available for Mixed11 addressing CanTp and is automatically activated when Mixed11 addressing is configured for that connection.

The addressing type parameter of a connection can be configured here:

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslAddressingType

Additionally, the following configuration switch must be enabled:

/Dcm/DcmConfigSet/DcmGeneral/DcmForeignDiagnosticRequestDetectionEnabled

8.24 How to Save RAM using Paged-Buffer for Large DIDs

8.24.1 Introduction

According to all up to now released AUTOSAR DCM SWS documents, the only service that supports paged-data reading is service *ReadDTCInformation (0x19)*. For any other data access services, i.e. service 0x22 (ReadDataByIdentifier), it is not possible to implement paged-buffer reading, without the need of fulfilling a lot of conditions and accepting implementation drawbacks and unnecessary risks.

So, if a large amount (measured in hundreds of bytes or even some kilobytes) of data must be carried out from the ECU, the DCM shall have at least one buffer that can handle the entire DID data. To avoid this in most cases unnecessarily RAM resource waste, the MICROSAR Classic DCM offers a concept for paged-data reading, described in detail further below.

8.24.2 Functionality

In order to provide a user-friendly method for getting data from the application using the paged-buffer concept, a non-AUTOSAR extension of the already available DataServices client-server port interface is required: *ReadData() (paged-data-reading)*.

The advantage of this concept is the great flexibility it offers:

- The application has full control of how many bytes to be transferred and for simple “memory copy only” implementations will be able to optimally fill up the response transmission buffer.
- Only a single function must be implemented, that handles the complete data transfer.
- The imported diagnostic description ODX/CDD will not be affected by these changes, since the ECU project implementer just chooses the kind of the data access, such as it could be made for direct access to NvM signals.
  - If the diagnostic description defines a DID with multiple large signals, for example four signals with 1000Byte each, for all those signals the new access type can be used and the DCM can still have a small diagnostic buffer.
- The concept is not defined by AUTOSAR but fits the AUTOSAR conventions.

- Either RTE C/S port or a callback to a complex device driver can be used.
- All DID related ECUC parameters are re-used if they are applicable for that concept.

There are also some possible drawbacks that must be considered when paged-DID reading feature is activated:

Reading data using the paged-buffer access, could lead to some unwanted effects:

- Sudden transmission interruptions for multiple DID requests on SID 0x22.
  - If the application generally has slow data access, then up to now, without the paged-data access, it had only caused some RCR-RP responses on the bus. With paged-buffer enabled read data access, a slow data provision could lead to transmission abortion by the TP if the N_as/N_cs are significantly shorter than the application data provision rate.
  - Limiting the maximum number of DIDs per service 0x22 request to one will avoid such interruptions but may also lead to a major deviation from the OEM diagnostic requirements.

8.24.3 Implementation Limitations

When paged-data access of a DID is intended to be used, there are still some limitations that must be considered:

- A DID, whose data shall be read via paged-data access, shall only support read operation. No paged-data access for writing or I/O control is possible. So only service 0x22 (ReadDataByIdentifier) may use it.
- For DIDs, accessible via service 0x2A (ReadDataByPeriodicIdentifier), paged-data access shall not be used.
  - Since those DIDs (0xF200-0xF2FF) are limited to only 7 bytes of data (on CAN) it also makes no real sense to apply this concept.
- Paged-data access cannot be used for DIDRanges (see *8.20 Handling with DID Ranges*).
- The support of DIDRanges and the paged-data access DIDs lead to contradictory concepts regarding the design of service 0x22 processor:
  - For paged-data access DID, the length of the DID shall be known prior reading the data and starting the positive response transmission.
  - For DIDRanges due to a lack of appropriate interfaces, the response data length is first known to DCM after the data reading has finished.

    Thus, it is not possible to have both DIDRanges and paged-data access DID within the same DCM configuration.
- Service 0x2C (DynamicallyDefineDataIdentifier) shall not be supported in DCM configurations with paged-data access DID.
- Paged-data access cannot be used together with OBD VIDs with dynamical length. That means when AR 4.2.2 and paged DID support are enabled, the callout for

reading VIDs *GetInfotypeValueData()* shall not modify the value of the length parameter DataValueBufferSize (see *8.31.1 Implementation Limitations* for details).

8.24.4 Usage

From application point of view, paged-data reading concept using the new DataServices port operation does not differ very much from the AUTOSAR data reading via an asynchronous port interface.

Any single return value of the new *ReadData() (paged-data-reading)* is described in details within its API description table.

For simplification reasons the following pictures show the DCM to application flow reading a single DID, consisting of a single data signal, that provides its content via paged-data access. The *ReadDataLength()* usage in the example is only to show that paged-DID signals can also have dynamic length.

The following scenarios are covered below:

- *Straightforward DID Paged-Data Reading*
- *Error Handling During DID Paged-Data Reading*

8.24.4.1 Straightforward DID Paged-Data Reading

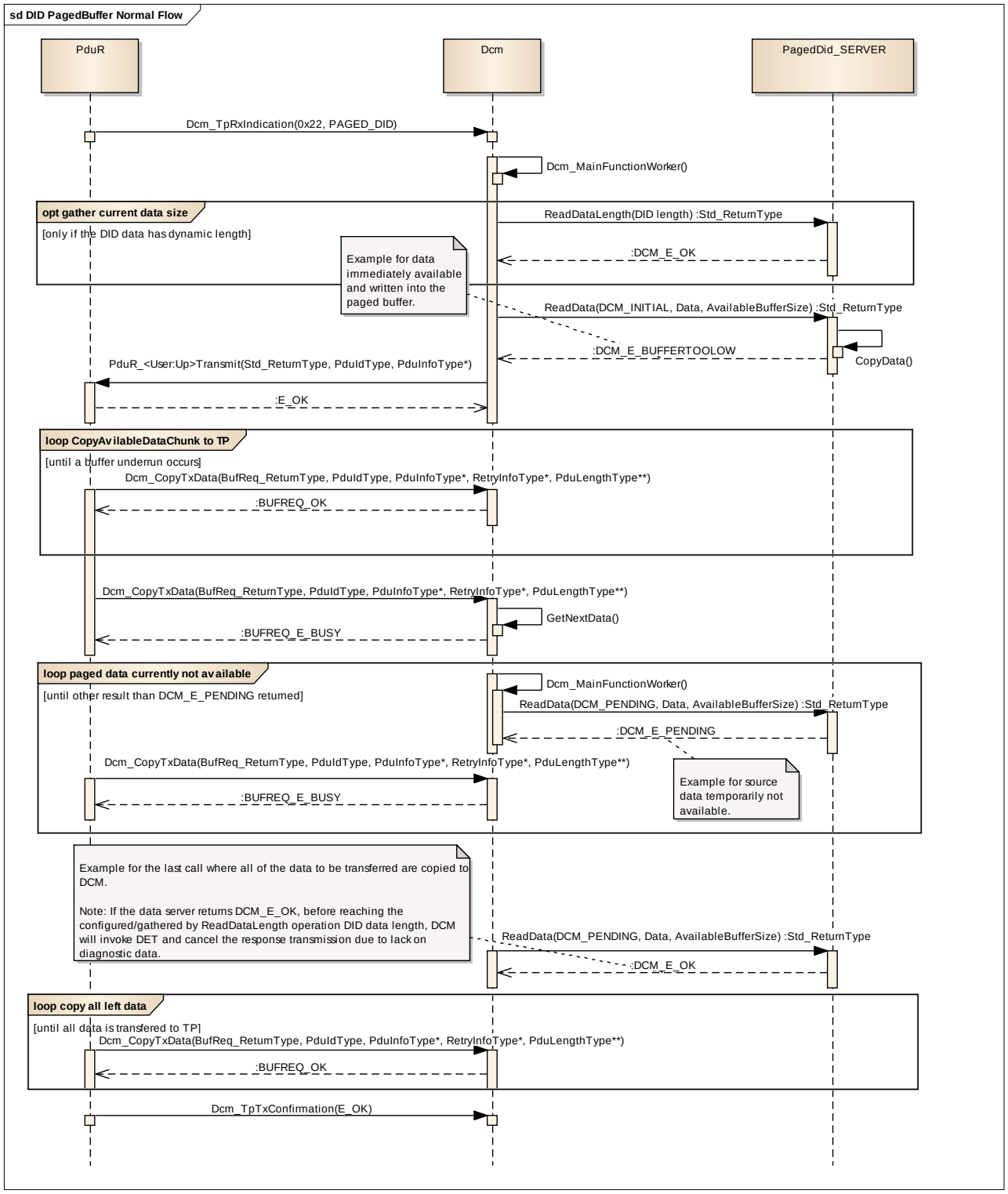

Figure 8-1 Straightforward DID paged-data reading

8.24.4.2 Error Handling During DID Paged-Data Reading

There are certain situations where the paged-data reading can be prematurely aborted:

> On response transmission abortion initiated by the TP layer caused by:

- Too slow data provision by the application, which lead to a N_as/N_cs timeout.
- Connection interrupted by the diagnostic client (i.e. no flow-control was sent).
- Other communication bus error has enforced the TP to abort the transmission.

- On protocol preemption via a higher priority client (e.g. OBD vs. UDS);
- On hitting RCR-RP limitation (if configured) caused by:
  - Too slow data provision by the application (over several seconds or even minutes).
  - Application deadlock that leads to an inability even to initiate the response transmission.

The figures below depict these situations and how the application is notified about the job interruption.

The common part is: The *ReadData() (paged-data-reading)* will be always called with OpStatus = DCM_CANCEL to notify the application that:

- it can initialize now any internal states (e.g. releasing semaphores),
- this is the last call of this data operation for current diagnostic service processing.

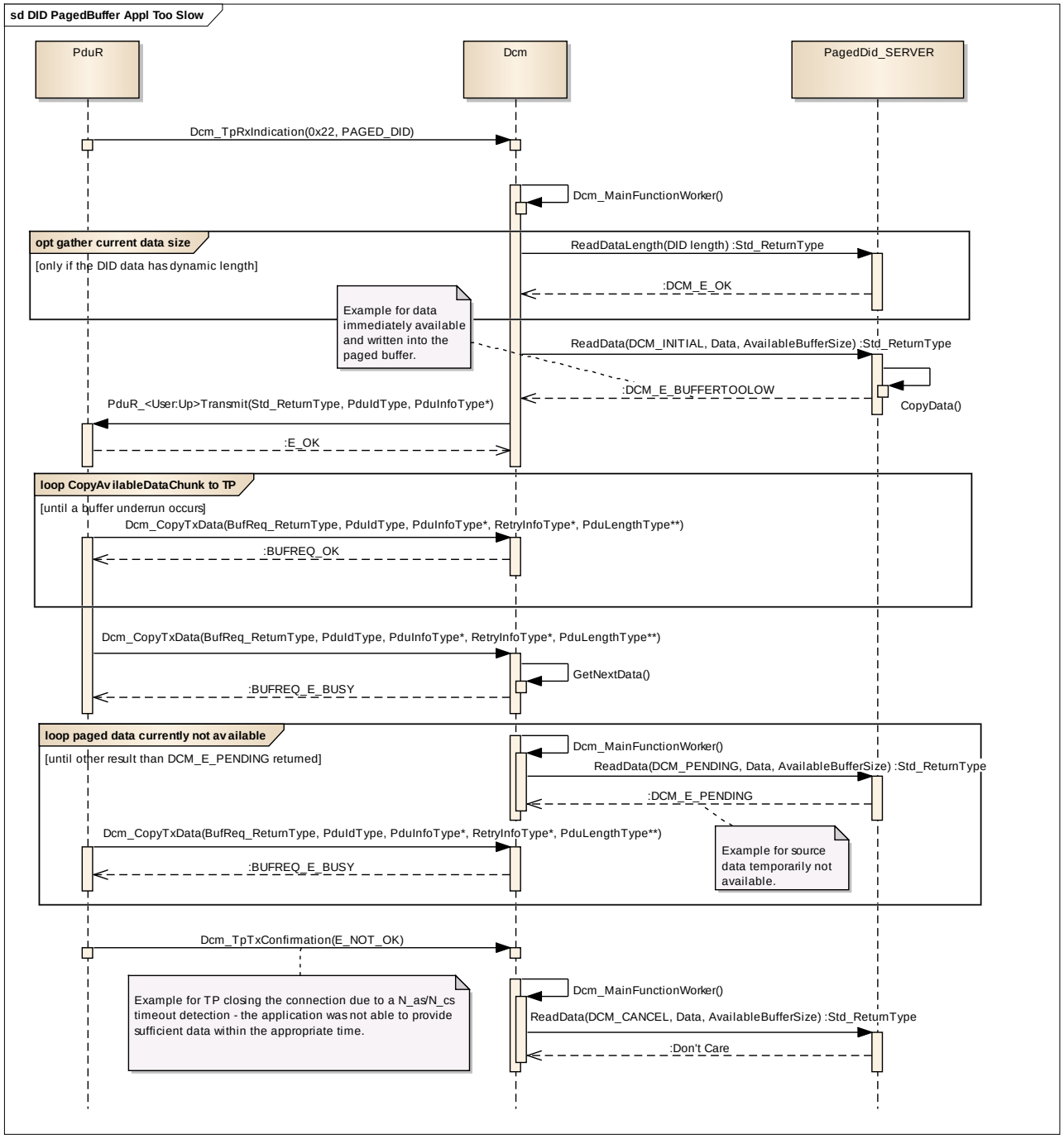

Figure 8-2 DID paged-data reading cancelled due to TP layer transmission abortion

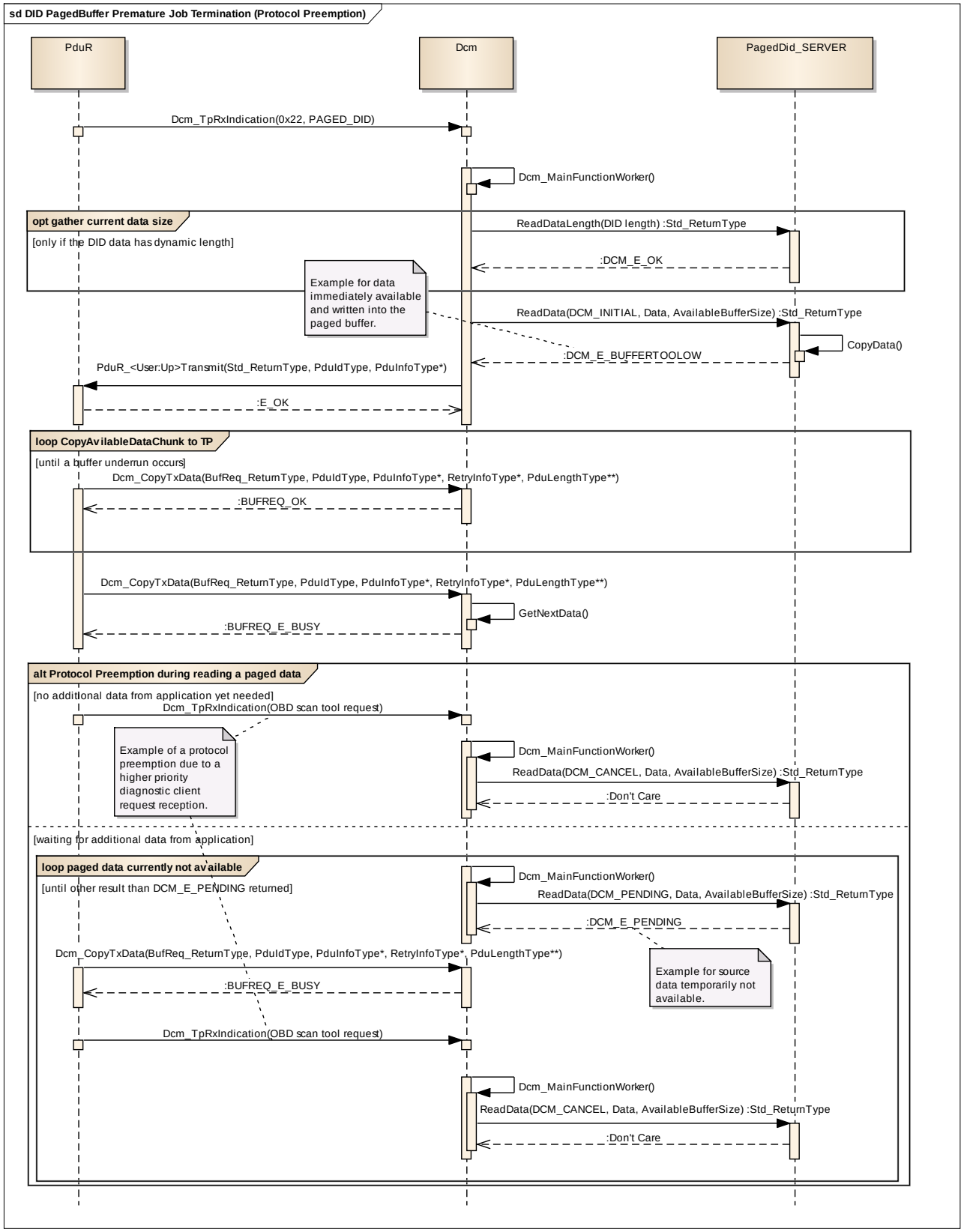

Figure 8-3 Protocol preemption during DID paged-data access

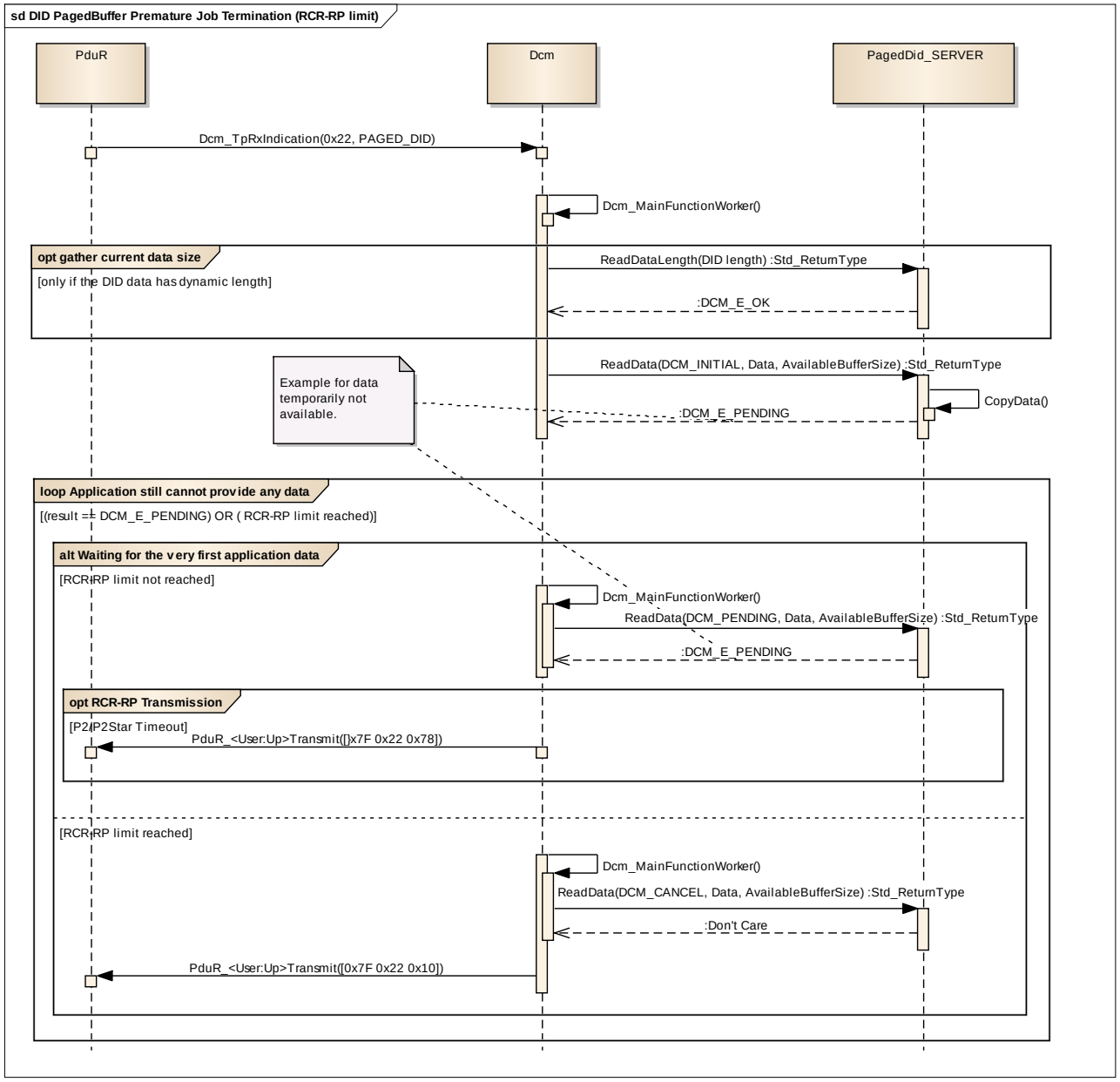

Figure 8-4 RCR-RP limit reached during DID paged-data access

8.24.5 Configuration Aspects

Note
The DCM parameter /Dcm/DcmConfigSet/DcmPageBufferCfg/DcmPagedBufferEnabled has no effect on the paged-data access of a DID. It affects only the paged-buffer support on service 0x19. In this way both services (0x19 and 0x22) can be independently configured for using paged-buffer data reading.

To configure a DID signal for paged-data access, the DCM BSWMD file must be changed in the following way:

Parameter: /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort

was extended by two new values:

- USE_PAGED_DATA_ASYNCH_CLIENT_SERVER – for SWC implementations
- USE_PAGED_DATA_ASYNCH_FNC – for callouts in ComplexDeviceDrivers.

From the parameters and containers already defined by AUTOSAR the following ones are only allowed to be used in context of a DID with paged-data access:

On DID level:

- /Dcm/DcmConfigSet/DcmDsp/DcmDspDid/DcmDspDidIdentifier
- /Dcm/DcmConfigSet/DcmDsp/DcmDspDid/DcmDspDidUsed
- /Dcm/DcmConfigSet/DcmDsp/DcmDspDid/DcmDspDidInfoRef
  - /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidRead – with all sub-parameters
- /Dcm/DcmConfigSet/DcmDsp/DcmDspDid/DcmDspDidSignal – with all sub-parameters

On DID Data level:

- /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort
- /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataConditionCheckReadFncUsed
- /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataConditionCheckReadFnc
- /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataReadFnc
- /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataSize
- /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataInfoRef
  - /Dcm/DcmConfigSet/DcmDsp/DcmDspDataInfo/DcmDspDataFixedLength

The size of the Paged-Buffer can be configured here (only used if implicit communication of the RTE is enabled – see 8.1.3 Exchanging data between OS tasks):

/Dcm/DcmConfigSet/DcmDsp/DcmDspReadPagedDataPageSize

Note
The value of /Dcm/DcmConfigSet/DcmDsp/DcmDspReadPagedDataPageSize shall be chosen with care. A small value leads to runtime overhead, because DCM must invoke the *ReadData() (paged-data-reading)* callout more often. On the other hand, the DCM internal buffer needs to be able to hold a full page before each *ReadData()* (paged-data-reading) callout, which may lead to a reduced utilization of the buffer.

8.25 How to Get Security-Access Level Specific Fixed Byte Values

8.25.1 Introduction

In some ECU projects it is desired, that some or all security-access level calculation algorithm shall use additional, level specific fixed bytes set to provide better flexibility and higher security protection. The latter is guaranteed by the split knowledge between provided implementation and project specific concrete values calculation.

Additionally, the diagnostic clients shall know these fixed bytes values, so in such cases these values are located within the diagnostic data exchange document (ODX/CANdela) imported by the system supplier into the MICROSAR Classic DCM configuration. In that way, both diagnostic client and server (ECU) have always the correct values.

To achieve this goal, MICROSAR Classic DCM extends the AR DCM standard ECUC configuration model by a new set of parameters (refer to the *Configuration Aspects*), as well as a new provided port operation *Dcm_GetSecurityLevelFixedBytes()*.

8.25.2 Usage

Once the fixed bytes are specified for the corresponding security levels, the DCM application implementer has the opportunity to access them within its software, by using the provided port operation *Dcm_GetSecurityLevelFixedBytes()*.

8.25.3 Security Level Fixed Bytes variant handling with VSGs

The DCM provides a means to define a set of security fixed byte values for a security level and to assign each fixed byte value to a Vehicle System Group. The required security fixed byte values can be enabled or disabled at run time of the ECU by enabling or disabling the corresponding Vehicle System Group.

The operation *Dcm_GetSecurityLevelFixedBytes()* will provide one of the enabled fixed byte values. If all fixed byte values of a security level are disabled, the operation will act as if there were no fixed bytes configured for this level.

For detailed information see also *8.34 Vehicle System Group Support.*

8.25.4 Configuration Aspects

If a security level shall provide a fixed byte set to the application, then the following container shall exist:

> /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow/DcmDspSecurityFixedBytes

For each fixed byte value, belonging to the set, an instance of the parameter below shall be specified:

> /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow/DcmDspSecurityFixedBytes/DcmDspSecurityFixedByteValue

FAQ

For the fixed bytes sets definition, the following rules do apply:

> It is allowed to define fixed byte sets only for some security-access levels

> It is allowed to have security-access level specific set size (e.g. one level with 5 bytes, another with 15)

> The order of creation of each byte value parameter within a set must be the same as the expected order of the values to be reported later to the application

8.26 How to Extend the Diag Keep Alive Time during Diagnostics

8.26.1 Problem Description

Per specification (*see* [1]) DCM shall keep the ECU alive (awaken) for a diagnostics reason under following circumstences:

> While in the default diagnostic session: As long as there is a diagnostic service in processing.

> While in a non-default session: As long as the DCM has not entered the default session again.

In some projects it is required that the ECU shall be kept alive for a certain time after the processing of a diagnostic request is finished. This leads to changes in the above listed situations as follows:

DCM will keep the ECU alive for a diagnostic reason when:

> While in the default diagnostic session:

  > as long as there is a diagnostic service in processing

  > **OR** for the time after the service processing is accomplished until the configured keep-alive time elapses.

> While in a non-default session:

  > as long as the DCM has not entered the default session again

  > **OR** as long as the running keep-alive timer is active. This condition is of course only applicable if the keep-alive time is configured to a value greater than the S3 time (set to 5000ms) since the keep-alive timer and the S3 timer are startet at the same time.

8.26.2 Configuration Aspects

If such an extended time for keep ECU alive is required, then please set up DCM in the configuration tool by specifying the keep-alive time in parameter:

/Dcm/DcmConfigSet/DcmGeneral/DcmKeepAliveTime

Per default a functionally addressed TesterPresent request with set SPRMIB (0x3E 0x80) actively extends the keep-alive time but does not start the keep-alive timer in case that the

default session was active and the keep-alive timer idle before. This means that such a TesterPresent request, in contrast to other diagnostic services, does not prevent the ECU from falling asleep when the ECU is in inactive diagnostic state. To alter this behavior, change the following parameter accordingly:

/Dcm/DcmConfigSet/DcmGeneral/DcmKeepAliveTimeStartOnFunc3E80

8.27 How to Recover DCM State Context on ECU Reset/Power On

8.27.1 Introduction

There are situations, where the ECU shall perform reset/power shutdown, but without losing some DCM internal states. Such states are for example:

- Active diagnostic session
- Active security access level (if applicable)
- The already managed communication control states (if applicable)
- Active state of control DTC setting (if applicable)
- Active state of any managed by DCM communication channel (DiagActive state)

Since this is not a feature supported by the AR standard per definition it was implemented in DCM for optional use only (refer to the configuration chapter below).

8.27.2 Functionality

In order to support the state context recovery, DCM has been extended by two new APIs for providing the data to be recovered on demand (*Dcm_ProvideRecoveryStates())* and to retrieve this data back on each reset /power on phase (*Dcm_GetRecoveryStates()*).

The data to be transferred is stored in the structure *Dcm_RecoveryInfoType*.

Caution
Please do always use both API to store and restore the context information. Only compatible versions of this data shall be used. Since the transferred data primarily consists of DCM internal data representation, it shall not be passed to DCM except if it was retrieved via the *Dcm_ProvideRecoveryStates()* API call.

On any state change (recovery data with default state does not have any effect), DCM will execute all notifications and actions related to that state transition. Due to this, DCM always executes the recovery process in the best applicable order for dependent states. For example:

- If security access and session change must be switched, then first the session change will apply then the security access level in order not to reset the security level during the session transition.
- If ControlDTCSetting shall be disabled and CommunicationControl shall apply too, then first the DTC setting will be disabled, and then the communication channels will change their states to avoid any unnecessary fault memory entries.

8.27.3 Configuration Aspect

If the recovery state feature is required for your project, please change the following parameter as described in its online help:

/Dcm/DcmConfigSet/DcmGeneral/DcmStateRecoveryAfterResetEnabled

8.28 How to Define a Diagnostic Connection without USDT Responses

Sometimes it may be necessary on a specific connection to suppress all kind of responses (positive or negative) in general. To configure such a connection, you must delete the following sub-container of it:

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolTx

8.29 How to Handle Multiple Diagnostic Service Variants

8.29.1 Introduction

DCM provides a means to execute filtering process on incoming requests at service, subservice, DID, RID, and DID operation level. For example, if a specific DID is configured in ECUC to be available for read and write purposes, the user can use DCM tools to update the configuration at runtime and to make the DID available only for read purposes. So, when a request comes with writing in that specific DID, the request will be rejected accordingly.

8.29.2 Filtering Level Availability and the Corresponding Filtering Tools

In the following two tables, namely, *Table 8-18* and *Table 8-19*, the filtering options available for each service are illustrated along with the corresponding filtering tools.

Service / Filtering Level	[All]	[0x22, 0x2A, 0x24, 0x2C, 0x2E, & 0x2F]	[0x31]	[0x23 & 0x3D]	[0x01 & 0x02]	[0x06]	[0x08]	[0x09]
Service	■							
Sub-service (Sub-function)	■							
DID		■						
DID Operation		■						
RID			■					
RID Operation			■					
Memory				■				
Memory Operation				■				
PID					■			
MID						■		
TID							■	
VID								■

Table 8-18 Filtering level availability

To get an advantage of DCM extended filtering tools, the extended filtering feature must be activated in the configuration tools. Refer to *Table 8-19* under column “Configuration Aspects” for more details.

Filtered Diagnostic Object	Filtering API / Callback	Configuration Aspects
Service, Sub-service (Sub-function)	Refer to: *5.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>* <Operation> = *Indication()*	Refer to: *8.6 How to Get Notified on a Diagnostic Service Execution Start and End*
DID, DID Operation	Refer to: *Dcm_FilterDidLookUpResult*	/Dcm/DcmConfigSet/DcmDsp/DcmDspDidLookUpFilterEnabled
RID	Refer to: *Dcm_FilterRidLookUpResult*	/Dcm/DcmConfigSet/DcmDsp/DcmDspRidLookUpFilterEnabled
RID Operation	Refer to: *5.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>* <Operation> = *Indication()*	Refer to: *8.6 How to Get Notified on a Diagnostic Service Execution Start and End*
Memory, Memory Operation	Refer to: *5.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>* <Operation> = *Indication()*	Refer to: *8.6 How to Get Notified on a Diagnostic Service Execution Start and End*
PID MID TID VID	Refer to: *8.29.3 Filtering OBD Objects*	Refer to: *8.29.3 Filtering OBD Objects*

Table 8-19 Filter diagnostic objects and the corresponding filtering APIs / Callbacks

FAQ
The filtering process is executed on already defined objects in the compile-time. The filtering process requires interference from the application. It is not possible that the application enables features via the filtering process in the runtime that is disabled in the first place in the compile-time. In case of OBD2, the application risks upon violation this rule a wrong reported “AvailabilityID” masks by DCM.

8.29.3 Filtering OBD Objects

In order to filter OBD objects and at the same time to report the appropriate “AvailabilityID” values in the most efficient way, the variant handling on OBD related objects is based on the feature *Calibration of Supported OBD Parameter Identifier* (refer to chapter *8.11.1.2)*. Since the “calibration” in this case is performed on-board, the calibratable data specified in the reference chapter shall be located in the **volatile memory (RAM)**. To change the calibration data memory location, please use the following parameter:

/Dcm/DcmConfigSet/DcmGeneral/DcmCalibrationOfObdIdsMemoryType

The concept requires that the application initializes the calibration data at every ECU power-on/reset, prior the call of the *Dcm_Init()* function. For that purpose, it is advisable for the application to keep prepared sets of the calibration data for each variant in its non-volatile memory and just copy it into the DCM volatile memory variant.

8.29.3.1 Suggested Preparation Methodology for Filtering Process of OBD Objects

In order to get a consistent content of these tables in the fastest way, we suggest you to follow the steps below:

- Create configurations (ECUC) files with DaVinci Configurator 5 for each variant you need. You will need only the configuration part of DCM, and only few mandatory BSWs which DCM refers to. These references will not be from importance for multiple-variant-handling, so they do not need to be maintained in future.
- Generate DCM configuration (Dcm_Lcfg.c/.h) for each of those variants.
- Copy the generated tables described in *Table 8-3 Calibratable OBD "availability parameter identifier" values* which exist in Dcm_Lcfg.c to your application.
- Rename the above copied tables according to the variant they belong to for better identification at the use time.
- If one variant includes one of the above-mentioned tables to be copied while the other does not (OBD service is disabled), make sure to add this table to your configuration anyway with zero entries.

8.30 How to Switch Between OBD DTR Support by DCM and DEM

Starting with AR version 4.1.1 DCM shall implement OBD MIDTID data retrieval for service *RequestOnBoardMonitorTestResults (0x06)* not directly from the application, but via a dedicated DEM API. Still, DCM provides a backward compatibility mode and if configured accordingly, it will handle the DTR values as before. Reading the following chapters, you will learn more about the impacts the new DTR value reporting implementation may have on your project. Then, if any choice is possible, you can decide which method you will prefer to use.

8.30.1 Implementation Particularities and Limitations

Once DCM is configured to provide DTR handling via DEM, any already available MID resp. MIDTID and MID DID (0xF6XX) in its configuration will be discarded. The configuration tool will inform you via "information" messages for all ignored related OBD MID parameters.

This does not mean that you must delete all these redundant data. Any available DID in range 0xF600-0xF6FF will be used as information for the DCM code generator that it is required a UDS MID mirroring of all the OBD MIDs. Since DCM does no more know which are the valid MID DIDs, it catches the whole DID 0xF6XX range for OBD MID reporting purpose.

This implies that:

> The UDS MIDs reported by DCM can only be those defined as MIDs under the DEM configuration. No application specific DIDs (i.e. some DIDs still to be read via C/S port) in the above cited range of identifiers is possible to be defined in DCM.

> Due to the DCM internal redirection of the MID DID handling to a DID range handler, the already known *Implementation Limitations* on *Handling with DID Ranges* do apply in this case too.

8.30.2 Configuration Aspect

Caution
The DCM configuration regarding the OBD MIDTID handling shall always be kept synchronized with the current DEM configuration.

> In case DCM is used together with the MSR DEM, it will notify you for any configuration mismatch by a corresponding error message, issued by an error directive at compile time (refer to *Table 9-1 Compile time error messages* for details on each message).

> In case another DEM vendor implementation is provided to the ECU project, a mismatching configuration between DCM and the DEM will result either in compile time errors (i.e. missing required DEM APIs) or may lead to an unexpected run time behavior as a result of the redundant and incompatible DEM and DCM MIDTID configurations (i.e. DEM does not support a certain MID, TID but DCM does support it or the DEM defines a different TID list for the same MID used within DCM etc.).

The OBD MIDTID handling is determined by setting the following DCM ECUC parameter accordingly: /Dcm/DcmConfigSet/DcmGeneral/DcmDtrDataProvisionViaDemEnabled

8.31 How to Enable Support of OBD VIDs with Dynamic Length

Depending on the DCM AR SWS compatibility mode, determined by the project license, the OBD VIDs will be retrieved from the application resp. DEM using corresponding variant of the *GetInfotypeValueData()* API. As you can see, the new API variant unconditionally (a project license is assumed as a constant property) provides a means for supporting a VID with variable data size. There is no additional configuration parameter to specify whether a certain VID shall have a variable length.

8.31.1 Implementation Limitations

While the VID reading via *RequestVehicleInformation (0x09)* is not really affected by the API change, *ReadDataByIdentifier (0x22)* does require some limitations to be taken into account, depending on the API variant. These limitations are of course only applicable if any OBD DIDs in the VID range (0xF800-0xF8FF) are to be supported by DCM.

The main difference in the usage of both API types is the point in time the DCM will calculate the final response length. When using API *GetInfotypeValueData()* in its AR 4.2.2 or newer variant, the final response length will be known to DCM ***_after_*** the VID data is read. This is the same situation as the one already known from chapter *8.20 Handling with DID Ranges* and therefore the *Implementation Limitations* regarding the DID length calculation do apply for these OBD VID DIDs too. Please note, that the maximum DID

length of those DIDs is determined by the corresponding VID data size parameter, as specified in *4.8.4 Configuration Aspects*.

8.32 How to setup DCM for Sender-Receiver Communication

Additionally to the *Client-Server Interface* type of communication with the application, also the Sender-Receiver (S/R) kind is supported for the following diagnostic services only:

- Data Identifier (DID) related:
  - Read Access:
    - *ReadDataByIdentifier (0x22)*
    - *ReadDataByPeriodicIdentifier (0x2A)*
  - Write Access:
    - *WriteDataByIdentifier (0x2E)*
  - IO Control:
    - *InputOutputControlByIdentifier (0x2F)*

The read and write S/R communication can be applied on a single DID data element or for the whole DID package as a single unit. The latter is required for the NvM SW-C communication to guarantee that all the data of a single NvM block is written consistently.

8.32.1 Implementation Limitations

When using the DCM S/R communication some limitations and particularities shall be considered:

- Please see *8.43.1 Data Types for DID Interfaces* for further details on the supported interface data types.
- The data element or DID shall have constant length.
- The data element or DID shall represent data that is synchronously accessible (the IO Control operation is an exception of this rule).
- For the DID related IOControl operations, the CEMR is limited by AR up to 4 bytes.
- If a DID supports any other operation than the above listed (i.e. *GetScalingInformation()*), those operations will be treated as if the data element was specified to have access of kind “SYNCH_FNC”. Therefore, a callout will be expected to be implemented by the application for the affected configuration object.

8.32.2 Application usage Scenario

To get S/R IOControl operation working with your application, the following design aspects shall be considered:

- On diagnostic request for service *InputOutputControlByIdentifier (0x2F)*

On each valid diagnostic request for an IO DID, DCM either delegates the IOControl job to the corresponding C/S port or performs multiple S/R port operation as a form of communication with the application. In the latter case if the requested IOControl operation is “ReturnControlToECU” DCM executes the same sequence of S/R port operations as for

the diagnostic session transition, described in the next section. The only difference is that not all IO channels of the IO DID will be reset, but only the ones, marked via the CEMR by the diagnostic client. For any other IOControl operation DCM will perform the following steps (per IO DID):

- If the operation was “ShortTermAdjustment” the “controlState” data will be updated with the content of the diagnostic request.
- The “controlEnableMask” will be updated with the content of the diagnostic request CEMR. (Please, carefully read the specifics of the CEMR handling in the corresponding chapter *InputOutputControlByIdentifier (0x2F)*).
- At last the “inputOutputControlParameter” will be set to the requested IOControl operation (e.g. DCM_SHORT_TERM_ADJUSTMENT), indicating that all related to this operation parameters are already set, and the operation can be executed.
- DCM starts waiting for the operation result (IOControlResponse). The wait state persists as long as the corresponding S/R has not yet been updated by the application, or DCM reads one of the values DCM_IDLE or DCM_RESPONSE_PENDING.
- Once DCM reads any other from the above-mentioned values (i.e. application has finished validation of the requested operation), the diagnostic service processing continues with:

If the result in IOControlResponse was DCM_POSITIVE_RESPONSE:

- The “underControl” will be updated by adding the requested bits from the CEMR.
- The “inputOutputControlParameter” will be set to DCM_IDLE, indicating to the application that the operation is now accomplished.
- DCM will now call the S/R port of the read operation to return to the client the actual IO DID values within the positive response.

In any other case for IOControlResponse, DCM will take the value as NRC for the initiated negative response that will follow.

- On diagnostic session transition to a session

Once DCM performs a diagnostic session transitions to the default session or to a non-default session where an IO DID under control is no longer supported, the “ReturnControlToECU” operation of the affected DID is executed. For the S/R IOControl DIDs the following steps will be performed (per IO DID):

- The “underControl” data will be updated with all bits set to zero, indicating no IO channel of this DID is under control.
- The “controlEnableMask” will be updated with all bits set, indicating all IO DID channels will be set back to normal mode.
- At last the “inputOutputControlParameter” will be set to 0x00 (i.e. DCM_RETURN_CONTROL_TO_ECU), indicating that all parameters related to this operation are already set and the operation can be executed.

Note
Since the IOControl operation “ReturnControlToECU” is a synchronous one that must always succeed, DCM will **not** expect any negative or pending response from the application via the IOControlResponse_<XX> S/R port. This is also the case when this operation is executed upon an explicit diagnostic client request.

This implies that the application shall not expect that for “ReturnControlToECU” the “inputOutputControlParameter” will be set to DCM_IDLE by DCM at a later point!

Caution
The following points must be considered during the implementation of S/R ports for IOControlRequest with operation “ReturnControlToECU”.

The application **must**

- always use the "underControl" data element of the corresponding IO-DID port to switch between real and overwritten value.
- not write any value to the "IOControlResponse" port in case the requested IOControl operation is "ReturnControlToECU", to avoid misinterpretation by DCM. This could occur while DCM is processing a new SID 0x2F request with operation other than the "ReturnControlToECU": DCM might interpret this as an acknowledgment to the new SID 0x2F request and confirm the request with a positive response.

8.32.3 Configuration Aspects

- To enable S/R communication on DIDs, you must specify the RTE usage on the corresponding DID data elements to be USE_DATA_SENDER_RECEIVER:

  /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort

- Additionally, if the S/R communication shall be applied on DID level for Combined Signal (Struct) or NV usage (i.e. all DID data elements will be merged into a single data block with the total length of the DID). For detailed information about Combined Signal and NV usage and the required configuration with the diagnostic transformer component DiagXf, see [22]. To enable S/R communication on DID level the following parameter shall be set accordingly:

  /Dcm/DcmConfigSet/DcmDsp/DcmDspDid/DcmDspDidUsePort

- To enable S/R communication via a PR-Port please set parameter:

  /Dcm/DcmConfigSet/DcmGeneral/DcmSenderReceiverPRPortsEnabled

For usage details of these parameters, please refer to the DaVinci Configurator5 online help.

8.33 How to Support Routine Info Byte with UDS RIDs

8.33.1 Introduction

The Routine Info Byte is a manufacturer specific value that is assigned to a routine and that can be reported to the tester when the diagnostic service *RoutineControl (0x31)* is requested. The DCM provides a means to report this Routine Info Byte without need of application intervention.

8.33.2 Configuration Aspects

If the DCM shall report the Routine Info Byte of a routine automatically, specify the value of the Routine Info Byte using following parameter:

/Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRoutineInfoByte

For every routine where this parameter is not supported, the application must provide the Routine Info Byte if needed.

8.34 Vehicle System Group Support

8.34.1 Introduction

Vehicle System groups (VSGs) is a multi configuration feature that provides a means to define sub-sets of diagnostic entities at configuration time that can be activated and deactivated at run time in the ECU. Deactivated entities will not be available at run time.

8.34.2 Functionality

A sub-set is defined by assigning a diagnostic entity to a VSG. A diagnostic entity can be assigned to one or several VSGs. The entity will be available at run time if at least one of the corresponding VSGs is enabled. Diagnostic entities that are not assigned to a VSG are part of the base variant and thus they are always available.

During initialization of DCM all configured VSGs will be disabled. The DCM application is responsible to enable all required VSGs after the initialization.

The base variant is always enabled and can not be disabled.

8.34.3 VSG operations

Beside the operations to enable and disable VSGs, the DCM also provides operations to request the current state of the VSGs:

- *5.2.2.8 Dcm_VsgSetSingle()*
- *5.2.2.9 Dcm_VsgSetMultiple()*
- *5.2.2.10 Dcm_VsgIsActive()*
- *5.2.2.11 Dcm_VsgIsActiveAnyOf()*

8.34.4 Configuration Aspects

All VSGs that shall be supported must be defined:

/Dcm/DcmConfigSet/DcmDsp/DcmDspVehicleSystemGroups

Following diagnostic entities can be assigned to the defined VSGs:

- Service

/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSidTabVehicleSystemGroupRef

- SubService

  /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService/DcmDsdSubServiceVehicleSystemGroupRef

- DID Operations (Read, Write, IO control)

  /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidRead/DcmDspDidReadVehicleSystemGroupRef

  /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidWrite/DcmDspDidWriteVehicleSystemGroupRef

  /Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl/DcmDspDidControlVehicleSystemGroupRef

- Memory access operations (Read, Write)

  /Dcm/DcmConfigSet/DcmDsp/DcmDspMemory/DcmDspMemoryIdInfo/DcmDspReadMemoryRangeInfo/DcmDspReadMemoryRangeVehicleSystemGroupRef

  /Dcm/DcmConfigSet/DcmDsp/DcmDspMemory/DcmDspMemoryIdInfo/DcmDspWriteMemoryRangeInfo/DcmDspWriteMemoryRangeVehicleSystemGroupRef

- RID

  /Dcm/DcmConfigSet/DcmDsp/DcmDspRoutineInfo/DcmDspRoutineAuthorization/DcmDspRoutineVehicleSystemGroupRef

- PID

  /Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidSvc01VehicleSystemGroupRef

- TID

  /Dcm/DcmConfigSet/DcmDsp/DcmDspRequestControl/DcmDspRequestControlVehicleSystemGroupRef

- MID

  /Dcm/DcmConfigSet/DcmDsp/DcmDspTestResultByObdmid/DcmDspTestResultObdmidTid/DcmDspTestResultByObdmidVehicleSystemGroupRef

- VID

  /Dcm/DcmConfigSet/DcmDsp/DcmDspVehInfo/DcmDspVehInfoVehicleSystemGroupRef

- Security Level Fixed Bytes (see also *8.25.3 Security Level Fixed Bytes variant handling with VSGs*)

  /Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow/DcmDspSecurityFixedBytes/DcmDspSecurityFixedByteVehicleSystemGroupRef

8.35 Usage Hints for Operation with SilentBSW

8.35.1 Introduction

The SilentBSW concept assures that a BSW module does not corrupt its own, the application or other BSW modules memory. The DCM module ensures that with:

> Additional runtime checks

> Assisted reviews by customer of the generated configuration output according to SafetyManual

> MICROSAR Classic Safe Silence Verifier (MSSV) tool

8.35.2 Configuration Aspects

Activate SilentBSW handling for DCM by setting the following DCM ECUC parameter accordingly: /Dcm/DcmConfigSet/DcmGeneral/DcmSafeBswChecks

Caution

> Please make sure that your delivery requires safety aspects.

> Please refer to the SafetyManual as well.

8.35.3 Automatically Enabled Features

The following table lists all automatically enabled features while DCM is in SilentBSW mode.

Feature	Condition	Description
Implicit RTE communication	Service 0x22 is configured and handled internally by DCM.	DCM performs additional size checks to cope with implicit RTE communication. Please refer to *8.1.3 Exchanging data between OS tasks* for more information.

Table 8-20 Automatically Enabled Features with SilentBSW

8.36 How to Support Diagnostic Service Dispatching

8.36.1 Introduction

In some ECU projects it may be necessary, that specific diagnostic services can be handled internally or externally at runtime.

The DCM provides a means to dispatch between internal or external diagnostic service processing with need of application intervention (callout).

8.36.2 Functionality

In order to support the service dispatching, DCM has been extended by an API *Dcm_HandleServiceExtern()*. This callout must be implemented by the application. It will only be called if the corresponding service is configured accordingly. For the other services there will be no application intervention. The callout is invoked after the SID specific checks have been done. Furthermore the descission of the application will be stored within each service processing.

8.36.3 Configuration Aspects

The service dispatching is determined by setting the following DCM ECUC parameter on internally supported diagnostic service level accordingly:
/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSidTabServiceDispatcher

Note
By enabling this feature all sub-services can still be called. This applies also for external sub-services.

8.37 How to provide an additional byte (specific cause code) to negative responses

8.37.1 Introduction

For detailed information about the cause of a negative response, DCM provides the possibility to add an additional specific cause code to negative responses.

8.37.2 Functionality

Additionally to the NRC, the application can register a specific cause code via API *Dcm_SetSpecificCauseCode()* during processing of a diagnostic request. In case of a negative response, DCM will append the registered specific cause code to the negative response of the current request.

8.37.3 Configuration Aspects

To enable the support of the specific cause code please enable the following parameter:
/Dcm/DcmConfigSet/DcmMiscellaneous/DcmSpecificCauseCodeEnabled

8.38 How to Deactivate S3 Timer

8.38.1 Introduction

Usually, the DCM supports a session timeout timer S3 Server, which keeps track of active non-default sessions. In some ECU projects it may be necessary, that the session timeout timer S3 Server is deactivated.

8.38.2 Functionality

DCM provides the possibility to deactivate the S3 timer functionality. Without active S3 timer in a non-default session, no session fallback into default session gets triggered.

8.38.3 Configuration Aspects

If no S3 timer is required, the following parameter must be disabled:
/Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmSessionTimerS3Enabled

8.39 How to Configure Generic Connections

8.39.1 Introduction

By default, DCM defines non-generic connections. The resources of a non-generic connection are reserved for and can be used only by one client (tester). The tester is statically defined in the pre-compiled time and is configured per main connection. The run-

time resources can be considerably decreased if the resources of one connection are shared by a pool of clients. The connection is generic when it shares its own resources among a pool of clients.

In case of generic connections, the tester information is not statically defined in the pre-compile time but dynamically provided by *Dcm_StartOfReception()* via meta data in the run time.

8.39.2 Functionality

Within a single configuration, both non-generic and generic connections may exist. Based on the meta data length of the Pdus reserved for a connection, the connection is attributed as generic or non-generic.

8.39.3 Configuration Aspects

- To configure a connection as generic, the Meta data lengths of the Pdus reserved by this connection must have values of 2 bytes.

  In other words, each

  /EcuC/EcucPduCollection/Pdu/MetaDataLength,

  of the Pdus referenced by,

  /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolRx/DcmDslProtocolRxPduRef,

  /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolTx/DcmDslProtocolTxPduRef, or

  /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslPeriodicTransmission/DcmDslPeriodicConnection/DcmDslPeriodicTxPduRef,

  must have a value of 2 bytes.

- The Ecu own address of a generic connection must be configured

  /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslPhysicalEcuAddress

- The following parameter is not used when a connection is generic:

  /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolRxTesterSourceAddr

8.39.4 Limitations

- Parallel processing of multiple generic testers of the same generic connection is not allowed. However, parallel processing of multiple generic testers each of distinct generic connection is allowed. E.g., if DCM processes a request of a generic tester and another request was admitted for a different generic tester affiliated to the same generic connection, DCM ignores the request of the second tester. However, if the second tester is affiliated to a different generic connection, the request is handeled/prioritized as usual.
- As restricted by [1], all Pdus affiliated to a generic connection must have the same length and all must have the same meta data length.
- Once a configuration is set to be post-build loadable, it shall be considered as generic-connection capable regardless the existence or the absence of generic connections in

this configuration. This restriction is set to protect DCM in case a generic-connection incapable configuration in pre-compile phase becomes generic-connection capable in post-build phase.

- An address of a client affiliated to a generic connection has a valid value range between 0x00 and 0xFF. The restriction is set as such because the client address of a generic connection is a run-time parameter conveyed to DCM via meta data which is a byte array. The client address of a generic connection must not be confused with the parameter DcmDslProtocolRxTesterSourceAddr which is the client address for a non-generic connection.

8.40 How to Persist Dynamic Defined DIDs

8.40.1 Introduction

DCM is capable of storing current Dynamic Defined DID data/states in non-volatile memory, so that they are available again after reset/initialization.

8.40.2 Functionality

The Dynamic Defined DID non-volatile data block used by the DCM must be configured to match the size of the underlying type. Since the actual size depends on compiler settings and platform properties, this size cannot be calculated by the configuration tool.

To find the correct data structure sizes, you can use temporary code to perform a ‘sizeof’ operation on the data types involved or check your linker map file if it contains this kind of data.

The MICROSAR Classic NvM implementation supports a feature called RAM-/ROM Block Size checks (refer to [16]) to verify the correct configuration of block sizes. It is strongly recommended to enable this feature; it also provides a very easy way to find out the correct block sizes.

NvRam Item	Type
Dcm_Svc2CNvMData	Dcm_Svc2CDynDidNvMDataType

Table 8-21 Persistent Dynamic Defined DID NvRam Block

Note
Enabling DcmDspDDDidClearOnStateChange can lead to deletion of Dynamic Defined DIDs after reset. This is because the DCM switches automatically to default session and evaluates all preconditions of Dynamic Defined DID (e.g. session or security levels). This applies only if DCM recovery feature (*8.27 How to Recover DCM State Context on ECU Reset/Power On)* is not activated.

Caution
Restored Dynamic Defined DID data will be deleted automatically and completely if configuration or, in case of PBS, active variant changes.

8.40.3 Configuration Aspects

To enable the persistence of the Dynamic Defined DIDs in DCM, the following configuration parameter must be set up:

/Dcm/DcmConfigSet/DcmGeneral/DcmDDDIDStorageBlockIdRef

Furthermore, the parameters NvMBlockUseSetRamBlockStatus and NvMSelectBlockForWriteAll shall be set to TRUE:

/NvM/NvMBlockDescriptor/NvMBlockUseSetRamBlockStatus,

/NvM/NvMBlockDescriptor/NvMSelectBlockForWriteAll

- The name of the RAM block data must be set to the symbol “Dcm_Svc2CNvMData“:

  /NvM/NvMBlockDescriptor/NvMRamBlockDataAddress

8.41 How to Authenticate

8.41.1 Using the DCM internal implementation

8.41.1.1 Introduction

Authentication is achieved via Service 0x29, through a two part process:

1) The tester must request 0x29 with sub-function Verify Certificate Unidirectional (0x01 - only the tester must be verified) or with sub-function Verify Certificate Bidriectional (0x02 – both the tester and the ECU must verify each other). If successful, the ECU responds with a positive response with authentication parameter “Certificate verified, Ownership verification necessary”.
2) If the above request is successful, the tester must request service 0x29 with sub-function Proof Of Ownership (0x03). With this request, the certificate role and white lists are read from the KeyM module, to be used in proceeding authentication checks. If this sub-function fails, the authentication process is incomplete, the ECU will fallback to deauthenticated state, and the request sequence must be started again.

Caution
Only the latest tester that verified the certificate (Unidirectional or Bidirectional) may ask for Proof Of Ownership, other testers will instead get a NRC 0x22. Example:
Tester 1 requests 0x29 with sub-function 0x01 or 0x02
Tester 2 requests 0x29 with sub-function 0x01 or 0x02
Tester 1 requests 0x29 with sub-function 0x03 -> NRC 0x22
Tester 2 requests 0x29 with sub-function 0x03 -> Positive response

8.41.1.2 White lists

Certificates used in the authentication process can contain four types of white lists:

- Service white list
- DID white list
- RID white list
- MemorySelection white list

The Service white list is used for authentication checking on both the DSD (Diagnostic Service Dispatcher) level and the DSP (Diagnostic Service Processing) level. Below is an example of a Service white list.

Example Service white list
0x17 0x12 0x21
0x22
0x22 0x12 0x46
0x2E 0x12 0x34
0x2F 0x12 0x34
0x2C 0x01 0x12 0x34
0x31 0x01
0x31 0x01 0x12 0x34
0x31 0x02 0x12 0x34
0x31 0x03 0x12 0x34

The above Service white list shown in red are Service white list elements that correspond to DID or RID access controls. Like the DID, RID, and MemorySelection white lists, these elements are processed in DSP.

Differentiation between DSP or DSD elements in Service white lists is as follows, an element belongs to the DSP if:

- element size == 3 && byte[0] == 0x22 || 0x2E || 0x2F
- element size == 4 && byte[0] == 0x31

DSP DID elements in Service white lists starting with 0x22, 0x2E, or 0x2F represent a DID operation and the DID identifier, meaning that 0x22 0x12 0x46 allows DID read operations for DID 0x1246. Similarly, elements starting with 0x2E and 0x2F correspond to write and IO Control respectively. These Service white list entries are handled like DID white list entries.

DSP RID elements contain the sub-function which represents the allowed RID operation (0x01 = Start Routine, 0x02 = Stop Routine, 0x03 = Request Result). E.g. 0x31 0x01 0x12 0x34 means to grant access RID with operation Start Routine for RID 0x1234. These Service white list entries are handled like RID white list entries.

DSD elements are simply compared to the first bytes of the incoming request on DSD level. If no matching element is found, the request is rejected.

Note
Service white list entry sub-function bytes are expected to **not** contain SPRMIB.

Note
White list checking is only necessary if the role verification denies access in the authenticated state on that level.

8.41.1.3 Persistence of Authentication States

DCM may persist the authentication state depending on configuration. If the referenced mode rule is activated, the authentication states are written to non-volatile memory if changed. This might occur if sub-function 0x03 or 0x00 of service 0x29 is processed. If the mode rule is deactivated, the persisted authentication states stored in the NvM are ignored upon reset. Furthermore, the authentication states are not written back to the NvM.

Caution
If a fallback of authentication state is triggered due to detection of an idle connection and the state transitions to deauthenticated, it is not persisted to non-volatile memory. Therefore, it is highly recommended to permanently deactivate the mode rule after End-of-Line. Otherwise, there might be a possible security leak.

> To enable persistence of authentication states, a mode rule shall be referenced via this parameter:
Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationPersistStateModeRuleRef

> For every authentication connection, a NvM block needs to be referenced where the authentication states shall be stored at. The reference to the NvM block shall be specified via this parameter:
Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationConnection/DcmDspAuthenticationPersistStateBlockIdRef

> The authentication states non-volatile data block used by the DCM for each connection must be configured to match the size of the underlying type. Since the actual size depends on compiler settings and platform properties, this size cannot be calculated by the configuration tool.

> To find the correct data structure sizes, you can use temporary code to perform a ‘sizeof’ operation on the data types involved or check your linker map file if it contains this kind of data.

> The MICROSAR Classic NvM implementation supports a feature called RAM-/ROM Block Size checks (refer to [16]) to verify the correct configuration of block sizes. It is strongly recommended to enable this feature; it also provides a very easy way to find out the correct block sizes.

NvRam Item	Type
Dcm_AuthMgrNvMData	Dcm_AuthMgrNvMDataType

Table 8-22 Persistent authentication states NvRam Block

8.41.1.4 Fallback of Authentication States

According to [3] DCM shall make a transition from authenticated into deauthenticated state for a configured connection if the following conditions apply:

> The DCM was in default session when the last diagnostic response was sent on that connection and

> DcmDspAuthenticationDefaultSessionTimeOut is configured and no valid diagnostic request was received on that connection for DcmDspAuthenticationDefaultSessionTimeOut seconds after the last Dcm_TpTxConfirmationon that connection.

It was not specified what shall happen if a tester is thrown out of any non-default session due to a request from another tester with higher priority or a call of *Dcm_ResetToDefaultSession()*. In such a scenario an authenticated tester would stay authenticated for ever, because a S3 timeout does not occur. Since the purpose of the feature is to detect and deauthenticate an idle connection, DCM is implemented in the following way: The fallback-timer is started independently of the current session with each response to a tester request. Depending on the configuration, the fallback-timer can be smaller or greater than the S3-timer (5 seconds). This means, that during non-default session always either the fallback-timer or the S3-timer will elapse. Either one of them will lead to deauthentication. In case that DCM transits into default session for example due to client prioritization, then DCM checks whether the fallback-timer is still running. If so, then the tester will stay authenticated until the fallback-timer is elapsed, if not, then the tester is considered to be idle and is deauthenticated. In case that a tester transits explicitly into default session via service 0x10, then the tester will of course not be deauthenticated.

The fallback-timer for the default-session can be configured here:

/Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationDefaultSessionTimeOut

Caution
If service 0x29 is configured to be allowed only in non-default session and an authenticated tester shall always be deauthenticated during transition into default session, DcmDspAuthenticationDefaultSessionTimeOut shall be set to zero.

8.41.1.5 Deauthentication via API

In some cases, it is desired to deauthenticate a tester on demand. Examples could be: on timeout of a self-implemented timer or the expiration of certificates. This can be done via the

API *Dcm_DeauthenticateConnection()*. But there are a few things that need to be considered:

- There is no restriction, when to call the API.
- Since there is no restriction, the API can be called during service processing of either an authentication request or any other request. The actual update of the authentication state is postponed until the end of this service processing. So, a tester might be deauthenticated although the prior authentication request was successful.
- If set via this API, the authentication state will not be persisted into the NvM. On startup, the application can decide via the ModeRule to not restore the authenticated state.

8.41.2 Using an external implementation

In some cases, it may not be possible to use the internal implementation of service 0x29 due to requirements which deviate from [10]. In this case, the DCM offers the option of processing the service completely externally. However, since it can be very complex and time-consuming to implement corresponding precondition checks, MICROSAR DCM offers the option of continuing to have these precondition checks internally. The following interfaces are offered for the purpose of setting the authentication state, role and whitelists by the application:

- *Dcm_DeauthenticateConnection()*
- *Dcm_AuthenticateConnection()*

To enable the interfaces the following configuration has to be followed:

- Service 0x29 has to be implemented externally /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService
- No subservice shall be configured /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSubService
- All connections which shall be authenticated, shall be referenced /Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationConnection/DcmDspAuthenticationConnectionMainConnectionRef

Please refer also to section *4.19 Authentication (0x29)* for general information.

8.42 How to Bypass Pre-Condition Checks

8.42.1 How to Support Bypass Mode for Security-Access

8.42.1.1 Introduction

In some ECU projects it is desired, that all security-access levels shall be unlocked on demand. DCM provides means for bypassing the security access via an API. This results in a common mechanism to disable security level checks in DCM.

8.42.1.2 Functionality

The security bypass mode can be enabled by using the provided port operation *Dcm_SetSecurityBypass()*. While the security bypass mode is active no security level

notifications will be called anymore. Furthermore, a diagnostic session change does not lock the ECU. The bypass mode is active until the API is called to disable the bypass mode or the DCM is reset/re-initialized.

Caution
Enabling the security bypass mode leads to an unlocked ECU with changed behavior of the security access (e.g. no notification on security level changes, diagnostic session changes, etc.). Therefore, the following points must be considered:

- Please be aware that this functionality disables security-access completely!
- To avoid security state inconsistencies the bypass mode shall only be applied if the security is locked.
- Also, it is forbidden to use the *Dcm_GetSecurityLevel()* API while this feature is active, since all levels are enabled.
- If the bypass mode shall be applied, the application shall prohibit (diagnostic request indication) any service 0x27 request until bypass mode is triggered.
- During bypass mode each security seed request is responded with zero seed since all security levels are unlocked. This implies, that service 0x27 state machine will never be changed on active bypass mode.
- Changing between diagnostic session with active bypass mode already active DIDs are not removed from scheduling unless it is not supported in the new diagnostic session. So please set the session restrictions of DIDs carefully when enabling this feature.
- Deactivating the bypass mode with active RTE notifications will report to the RTE a security level change from locked state to locked state.

8.42.1.3 Configuration Aspects

If a security bypass mode is required, then please set up DCM in the configuration tool by specifying the security bypass in parameter:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityBypassEnabled

8.42.2 How to Support Bypass Mode for Authentication

8.42.2.1 Introduction

It’s required in some cases to disable the authentication roles checking, for example in the development process when the OEM doesn’t want to use authentication certificates in the development phase. DCM provides a way to bypass the authentication roles checking via an API.

8.42.2.2 Functionality

The authentication bypass feature can be enabled by using the provided port operation *Dcm_SetAuthenticationBypass()*.

- While the authentication bypass mode is enabled clients are still allowed to be authenticated/de-authenticated.
- The normal operation of service 0x29 (e.g. storing data into non-volatile memory, reading white lists from KeyM etc.) is not affected by the bypass feature.
- The bypass mode information is not stored in NvM.

> Authentication state change notifications are not affected as well.

8.42.2.3 Configuration Aspects

If an authentication bypass mode is required, then please set up DCM in the configuration tool by specifying the authentication bypass in parameter:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspAuthenticationBypassEnabled

8.43 Supported Data Types for DIDs and RIDs

DCM supports typed data interfaces for the communication with the application according to [3]. This is available for data use ports Client-Server Interface, Sender-Receiver Interface, and DCM function callout Interface as well as for all supported routine operations. Please consider these general cautions regarding all types of typed interfaces.

Caution
Multi-byte array data types (like Uint16_N) with many elements can lead to high RAM/stack consumption.

Caution
Please note, that float values are exchanged between a diagnostic client and the application without considering the underlying representation (in hardware/memory). DCM will essentially copy the memory contents considering only the endianness.

Caution
The diagnostic client may exchange any sort of NaNs or other special floating-point values (e.g. infinities) with the ECU. Since DCM just copies the memory contents from the data bus, interprets them as floating-point values and passes them to the application or vice versa, the handling of the potentially caused exceptions according to IEEE 754 must be done by the application.

8.43.1 Data Types for DID Interfaces

Typed interfaces for DIDs are supported for the following services only:

> Read Access:
>> *ReadDataByIdentifier (0x22)*
>> *ReadDataByPeriodicIdentifier (0x2A)*

> Write Access:
>> *WriteDataByIdentifier (0x2E)*

> IO-Control (partially):
>> *InputOutputControlByIdentifier (0x2F)*

When using the DCM communication with typed interfaces some limitations and particularities shall be considered:

> If a DID supports any other operation than the above listed (i.e. *GetScalingInformation()*), those operations will be treated as if the data element has the data type Uint8.

By default, DCM uses the interface data type based on the *DcmDspDataType* selection as displayed in *Table 8-23* and *Table 8-24*. However, in previous versions the generated data type was always Uint8[n] for specific types. To restore this behavior for better backward compatibility, DCM offers the option to overwrite the default on two different levels:

> Global: Select the default setting for all data elements which support this feature: /Dcm/DcmConfigSet/DcmGeneral/DcmTypedDataInterfacesEnabled

> Local: Overwrite the default setting for single data elements: /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataTypedDataInterfaceEnabled

For usage details of these parameters, please refer to the DaVinci Configurator 5 online help.

8.43.1.1 Sender-Receiver Interfaces

The following Sender-Receiver interfaces are provided based on the selected data type per data element for the element specific interface use case. Data elements with dynamic length are not supported for Sender-Receiver interfaces in general.

DcmDspDidUsePort:		
USE_DATA_ELEMENT_SPECIFIC_INTERFACES		
DcmDspDataUsePort:		
USE_DATA_SENDER_RECEIVER		
DcmDspDataType	**Service 0x22, 0x2E, 0x2A**	**Service 0x2F**
BOOLEAN	boolean	boolean
UINT8	uint8	uint8
UINT8_N	uint8[n]	uint8[n]
UINT16	uint16	uint16
UINT16_N	uint16[n]	uint8[n]
UINT32	uint32	uint32
UINT32_N	uint32[n]	uint8[n]
SINT8	sint8	-[17]
SINT8_N	sint8[n]	-[17]
SINT16	sint16	-[17]
SINT16_N	sint16[n]	-[17]
SINT32	sint32	-[17]
SINT32_N	sint32[n]	-[17]
FLOAT	float32	-[17]
FLOAT_N	float32[n]	-[17]

Table 8-23 Supported Sender-Receiver interface data types

Additionally, DCM supports operations on DID level, where the whole DID package is treated as a single unit (*DcmDspDidUsePort* set to either USE_ATOMIC_SENDER_RECEIVER_INTERFACE_AS_SERVICE or USE_ATOMIC_NV_DATA_INTERFACE). This is required for the NvM SW-C communication to guarantee that all the data of a single NvM block is written consistently.

The combined data type consists of all the configured data elements referenced by the DID. The final data type (combined) will be provided by the diagnostic transformer component DiagXf. For more information about the data types of the members please check the DiagXf documentation.

8.43.1.2 Client-Server and Function Callout Interfaces

The table shows the supported data types for Client-Server and function callout interfaces. In case of data elements with dynamic length, the interface limited to Uint8[n] array.

[17] Not supported and indicated by generator validation rule.

DcmDspDidUsePort:
USE_DATA_ELEMENT_SPECIFIC_INTERFACES
DcmDspDataUsePort:
USE_DATA_ASYNCH_CLIENT_SERVER or USE_DATA_SYNCH_CLIENT_SERVER or USE_DATA_ASYNCH_FNC or USE_DATA_SYNCH_FNC

DcmDspDataType	Service 0x22, 0x2E, 0x2A	Service 0x2F
BOOLEAN	boolean	uint8[n]
UINT8	uint8	uint8[n]
UINT8_N	uint8[n]	uint8[n]
UINT16	uint16	uint8[n]
UINT16_N	uint16[n]	uint8[n]
UINT32	uint32	uint8[n]
UINT32_N	uint32[n]	uint8[n]
SINT8	sint8	uint8[n]
SINT8_N	sint8[n]	uint8[n]
SINT16	sint16	uint8[n]
SINT16_N	sint16[n]	uint8[n]
SINT32	sint32	uint8[n]
SINT32_N	sint32[n]	uint8[n]
FLOAT	float32	uint8[n]
FLOAT_N	float32[n]	uint8[n]

Table 8-24 Supported Client-Server and Function Callout interface data types

8.43.2 Data Types for Routine Interfaces

DCM supports the data types displayed in *Table 8-25* for routine signals. Based on the *DcmDspRoutineSignalType* selection DCM will generate the Client-Server or function callout interface with parameters of the data types listed below.

DcmDspRoutineSignalType	Interface parameter data type
BOOLEAN	boolean
UINT8	uint8
UINT8_N	uint8[n]
UINT16	uint16
UINT16_N	uint16[n]
UINT32	uint32
UINT32_N	uint32[n]
SINT8	sint8
SINT8_N	sint8[n]
SINT16	sint16
SINT16_N	sint16[n]
SINT32	sint32
SINT32_N	sint32[n]
FLOAT	float32
FLOAT_N	float32[n]
VARIABLE_LENGTH	uint8[n]

Table 8-25 Supported data types for RIDs

8.44 How to Configure DIDs with static content

8.44.1 Introduction

The DCM supports the handling of DIDs with static content without the need to interact with the application.

8.44.2 Functionality

The static content can be provided separately for a single data element. Therefore, the mixture of static and dynamic content within one DID is possible. When setting the static content for a data element then there will be neither a port nor a callout function required. In case of the specified data size is larger than required for the static content the DCM will extend the content with padding byte(s) specified in

/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataConstantValuePaddingPattern

8.44.3 Limitations

When using static content for data elements the following limitations shall be considered:

> Only read access is supported for data elements with static content.

> For static content of type decimal, the content is always reported in big endian byte order.

Note
For static content of type ASCII or HEX the endianness is not relevant because DCM interpretes the content as byte array.

8.44.4 Configuration Aspects

To set static content for a data element the following parameters need to be considered:

> Set the static content of the data element:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataConstantValue

> Select the type of the provided content (ASCII, decimal, …):
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataConstantValueType

> Specifies the padding byte(s) in case the specified data size is larger than required for the static content:
/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataConstantValuePaddingPattern

8.45 How to report security events to IdsM

8.45.1 Introduction

Intrusion Detection System Manager (IdsM) is used to report on-board security events. Those security events are triggered from basic software modules in case of malicious activities or violations. This can help in detecting software attacks or certain non-behaivoral pattern. Therefore, the DCM should report the security events that are related to its own component as mentioned in [4].

8.45.2 Functionality

The supported security events are listed in the following table:

Security event	Sid	Definition
DIAG_SEV_ECU_RESET	0x11	ECU has been successfully reset by sub-functions 0x01, 0x02 or 0x03.
DIAG_SEV_CLEAR_DTC_SUCCESSFUL	0x14	DTC information has been successfully cleared.
DIAG_SEV_ECU_UNLOCK_SUCCESSFUL	0x27	Successfully unlocked a security level.
DIAG_SEV_COMMUNICATION_CONTROL_SWITCHED_OFF	0x28	ECU communication was successfully switched off by sub-functions 0x01, 0x02 or 0x03.
DIAG_SEV_AUTHENTICATION_SUCCESSFUL	0x29	Successfull authentication.
DIAG_SEV_CERTIFICATE_FAILURE	0x29	Invalid certificate authentication (NRC 0x50-0x58). Note: the security event will be triggered independently of whether the NRC is replaced by /Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationGeneralNRC.

DIAG_SEV_NUMBER_OF_FAILED_AUTHENTICATION_ATTEMPTS_EXCEEDED	0x29	Not supported.
DIAG_SEV_WRITE_DATA	0x2E	DID has been successfully written.
DIAG_SEV_WRITE_INV_DATA	0x2E	Tester writes invalid data with service WriteDataByIdentifier. This event is triggered only when the application returns E_NOT_OK with ErrorCode 0x31. Note: only this event is reported (request out of range security event will not be reported).
DIAG_SEV_REQUEST_DOWNLOAD	0x34	Request download service (0x34) was successfully processed.
DIAG_SEV_DTC_SETTING_SWITCHED_OFF	0x85	DTC setting was successfully switched off using sub-function 0x02.
DIAG_SEV_SERVICE_NOT_SUPPORTED	-	NRC 0x11 (serviceNotSupported) or NRC 0x7F (serviceNotSupportedInActiveSession) was returned.
DIAG_SEV_SUBFUNCTION_NOT_SUPPORTED	-	NRC 0x12 (subFunctionNotSupported) or NRC 0x7E (subFunctionNotSupportedInActiveSession) was returned.
DIAG_SEV_INCORRECT_MESSAGE_LENGTH_OR_FORMAT	-	NRC 0x13 (incorrectMessageLengthOrInvalidFormat) was returned.
DIAG_SEV_REQUEST_SEQUENCE_ERROR	-	NRC 0x24 (requestSequenceError) was returned.
DIAG_SEV_REQUEST_OUT_OF_RANGE	-	NRC 0x31 (requestOutOfRange) was returned.
DIAG_SEV_SECURITY_ACCESS_DENIED	-	NRC 0x33 (securityAccessDenied) was returned.
DIAG_SEV_REQUESTED_ACTIONS_REQUIRES_AUTHENTICATION	-	NRC 0x34 (authenticationRequired) was returned.
DIAG_SEV_SECURITY_ACCESS_INVALID_KEY	-	NRC 0x35 (invalidKey) was returned.
DIAG_SEV_SECURITY_ACCESS_NUMBER_OF_ATTEMPTS_EXCEEDED	-	NRC 0x36 (exceedNumberOfAttempts) was returned.
DIAG_SEV_SECURITY_ACCESS_REQUIRED_TIME_DELAY_NOT_EXPIRED	-	NRC 0x37 (requiredTimeDelayNotExpired) was returned.

Table 8-26 IdsM Security Events definitions

Note
Only one IdsM security event is reported per request.

8.45.3 Configuration Aspects

> The IdsM security events which shall be reported have to be configured in the following container:
/Dcm/DcmConfigSet/DcmGeneral/DcmSecurityEventRefs

8.46 How to Support Secure Coding Features

8.46.1 Introduction

Secure coding is a feature offered by DCM to allow the data on the ECU to be modified in a secured manner. The purpose of the feature is to enable the manufacturer to control the type of data that is allowed to be written into the ECU. In principle, only signed data can be accepted and written into the ECU.

8.46.2 Functionality

The secure coding feature consists of the following 4 steps:

1. Transmission of a certificate into DCM
2. Writing the required data into the DCM buffer
3. Transmission of the signature of the written data
4. Updating the data in the application

In the first step, a valid certificate is transmitted to DCM and KeyM validates the received certificate. To begin the step, the tester sends a *Authentication (0x29)* sub-function 0x04 request to DCM with a valid certificate attached.

After the first step is successfully completed, the tester can start the second step by sending the data using a *WriteDataByIdentifier (0x2E)* request. At this stage, the data retrieved is stored in a buffer within DCM.

Note
If a piece of data is written at this stage, but the subsequent steps of the secure coding feature have not yet been conducted, the updated value of the data is not yet forwarded to RTE. If a *ReadDataByIdentifier (0x22)* request for this specific piece of data is received, DCM still responds the tester with the un-updated value of the data.

After the required data are written into the DCM buffer, the tester can send a *RoutineControl (0x31)* request with the signature of the data attached to start the routine to trigger step 3. Once DCM has received the signature of the data, it starts the authentication of the data stored in the buffer.

If the data in the buffer can be successfully authenticated, DCM triggers automatically the final step of the process, i.e. the data stored in the buffer is written to the application using the same procedure as *WriteDataByIdentifier (0x2E)*.

8.46.3 Configuration Aspects

In order to enable the secure coding feature in DCM, the following settings are required:

- *WriteDataByIdentifier (0x2E)* and *RoutineControl (0x31)* must be enabled.
- *Authentication (0x29)* sub-function 0x04 must be enabled.
- The certificates for data authentication must be defined in the following container: /Dcm/DcmConfigSet/DcmDsp/DcmDspAuthentication/DcmDspAuthenticationTransmitCertificate
- A Csm job for authenticating the secure coding data must be referenced in the parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmSecureCoding/DcmSecureCodingAuthenticationJobRef

Note
The referenced Csm job must trigger *Dcm_CsmSecureCodingValidationFinished()* after the validation is finished to report the result to DCM. Otherwise, the secure coding process cannot continue.

- The size of the secure coding buffer must be defined in the parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmSecureCoding/DcmSecureCodingDataBufferSize
- The reference to the routine that triggers the authentication and the write operation of the buffered data must be defined in the parameter: /Dcm/DcmConfigSet/DcmGeneral/DcmSecureCoding/DcmSecureCodingValidationRoutine
- The list of data identifiers, which are protected by the secure coding process, must be defined in the container: /Dcm/DcmConfigSet/DcmGeneral/DcmSecureCoding/DcmSecureCodingDataIdentifier

8.47 How to Use Diagnostic Service Pre-/Post-Handler

8.47.1 Introduction

In some cases, it is useful or necessary to alter states, fulfill preconditions or clean up before or after a diagnostic service is processed or just be informed about the request/response message data. This can be achieved in the application by the Pre-/Post-Handler function callouts.

8.47.2 Functionality

The Pre-/Post-Handler function callouts (see section *5.5.2.3* and *5.5.2.4* for details) are, in contrast to the manufacturer/supplier notifications, asynchronous callouts, which are called right before or right after the actual diagnostic service is processed. They are called when the diagnostic request itself is already accepted by the DCM or before the final response is sent. This means, that the Pre-Handler function callout will not be called when the service processing is denied by a manufacturer/supplier indication function. But the Pre- and Post-Handler function callouts will be called regardless of whether the diagnostic service is

implemented internally or externally. They are also called when Service Dispatching (see section *8.36*) is activated.

As soon as the Post-Handler is finished, the DCM issues, if necessary, the final response. For some diagnostic services, but not all, internal states are altered **only** if the successful transmission of a positive response was confirmed by the communication layer. New states will then be applied in the so-called post-processing phase. If no response was required, this phase begins immediately after the Post-Handler phase. See also *Table 8-27* for further information which services alters internal states during service processing or the post-processing phase.

The response, as indicated in the response data, might not be issued on the data bus in case of e.g.:

- SPRMIB (SuppressPositiveResponseMessageIndicationBit) is set.
- NRC 0x31 (RequestOutOfRange) is issued in combination with functional addressing.
- No TxPduId was configured for the connection.

Figure 8-5 shows the sequence diagram. As shown,

- if the Pre-Handler returns E_NOT_OK, the service processor will not be called and a negative response will be sent.
- f the service processor returns DCM_E_NOT_OK, the Post-Handler will not be called and a negative response will be sent.
- if the service processor returns DCM_E_OK it is still possible that a negative response is sent, when the Post-Handler returns E_NOT_OK.
- in case of a cancellation, the sequence will be aborted. This means, if the sequence is cancelled during service processing, the Post-Handler will not be called.

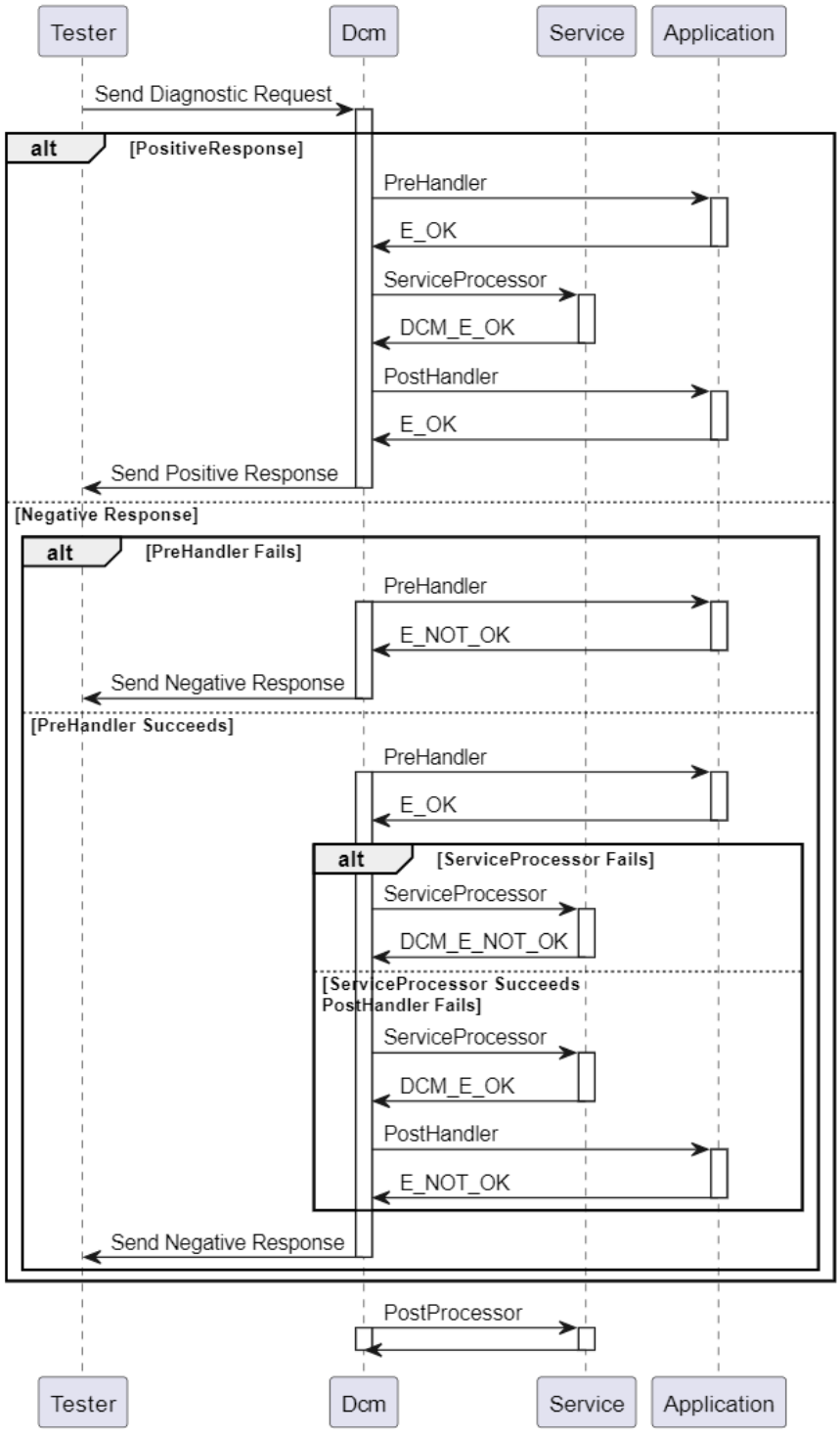

Figure 8-5 Pre-/Post-Handler Sequence

Caution

If the diagnostic service request is a functional Tester Present (0x3E 0x80) request, the Pre-/Post-Handler callouts will not be called.

Caution
Pre-/Post-Handler callouts will not be serialized by DCM in case that parallel service processing is enabled (see chapter *8.12.2 Diagnostic Client(s) Processing Parallelization* for more details). This gives the application full control over the degree of parallelization but must be considered during the design phase. The function parameters `ProtocolType`, `TesterSourceAddress` and/or `ConnectionId` can be used to distinguish between calls of the same Pre-/Post-Handler callout from different threads respectively protocols.

Alteration in Service-Processor	Transition in Post-Processor
Service 0x11	Service 0x10
Service 0x29	Service 0x27
Service 0x86	Service 0x28
	Service 0x29
	Service 0x85

Table 8-27 Point of time of state transition

8.47.3 Configuration Aspects

The Pre- and Post-Handler function callout can each be configured individually for every service. It is possible to configure the same function for more than one service. And it is even possible to configure the same function for a Pre- and a Post-Handler.

The function name for a Pre-Handler callout can be configured in the following container:

/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSidTabPreHandler

The function name for a Post-Handler callout can be configured in the following container:

/Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService/DcmDsdSidTabPostHandler

8.48 How to Use Protocol specific restriction for DIDs and RIDs

8.48.1 Introduction

The largest DID / RID used in the configuration determines the buffer size. In case of multiple protocols are required for tester prioritization, each protocol needs an own buffer. This can lead to high RAM consumption, as each buffer must be at least as large as the largest DID / RID. With “Protocol specific restriction” for DIDs and RIDs, it is possible to restrict the access to DIDs / RIDs on protocol level. The largest DID / RID accessible for a specific protocol determines the size of the buffer associated to the protocol.

The example in the following table shows the effect of the “Protocol specific restriction”. Without “Protocol specific restriction” the buffer size for each buffer must be 2000 bytes and the total RAM consumption is therefore 6000 bytes. The largest DID accessible (marked with x) in Protocol 1 is DID 0x1000 with a size of 2000 bytes. The needed buffer size for Protocol 1 is 2000 bytes. The largest DID accessible in Protocol 2 is DID 0x1001 with a size

of 500 bytes. Therefore, the needed buffer size for Protocol 2 is 500 bytes. Similar for Protocol 3 the largest RID is 0x2000 with a needed buffer size of 1000 bytes. The total RAM consumption is now reduced to 3500 bytes.

DID / RID	Protocol 1	Protocol 2	Protocol 3
DID 0x1000 (2000 bytes)	x	-	-
DID 0x1001 (500 bytes)	x	x	x
DID 0x1002 (700 bytes)	x	-	x
RID 0x2000 (1000 bytes)	x	-	x
RID 0x2001 (100 bytes)	x	x	x
RID 0x2002 (250 bytes)	x	x	x
Buffer size for Protocol	**2000 bytes**	**500 bytes**	**1000 bytes**

Table 8-28 Example for Protocol specific restriction

Please note, all byte sizes given above only refer to the payload. The also needed SID, DID and RID are not considered in the example above.

8.48.2 Functionality

The restriction can be individually configured for each DID / RID on protocol level (see Configuration Aspects below). If a DID / RID is requested from a tester, which is assigned to a restricted protocol, the NRC 0x31 (requestOutOfRange) is sent.

8.48.3 Limitations

> The protocol specific restrictions for DIDs and RIDs are meant to be invariant. For a detailed description refer to *8.19.1.2.1 Handling of State Execution Preconditions of Variant Diagnostic Entities*.

> This feature is not available in post-build loadable configurations.

8.48.4 Configuration Aspects

The Protocol restriction can be configured individually for every DID with the following parameters:

Read Access:

/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidRead/DcmDspDidReadProtocolRef

Write Access:

/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidWrite/DcmDspDidWriteProtocolRef

IO Control:

/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl/DcmDspDidControlProtocolRef

The Protocol restriction can be configured individually for every RID with the following parameter:

/Dcm/DcmConfigSet/DcmDsp/DcmDspRoutineInfo/DcmDspRoutineAuthorization/DcmDspRoutineProtocolRef

Caution
The above-mentioned parameters are used as white lists. This means that if access to a DID / RID should be allowed from a specific protocol, the protocol reference must be entered. Only if no restriction for the DID / RID is required at all, the parameter can be left empty.

8.49 How to prevent a security reset at a session self-transition

8.49.1 Introduction

Normally, the DCM resets the security level back to the “locked” level as soon as the session is changed. Some projects make it necessary for the Dcm not to reset the security level if the current active non-default session is requested again.

8.49.2 Functionality

During the session transition the DCM will check whether the current and the newly requested session are the same (e.g. transition from Extended → Extended session). If this is the case and this feature is globally enabled the security level will remain unchanged.

8.49.3 Configuration Aspects

If security levels shall remain unchanged after a self-transition in a non-default session, set the following parameter to true:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRetainOnSessionSelfTransition

9 Troubleshooting

9.1 Compile Error Messages

This chapter describes the error situations the DCM code checks and catches at compile time.

Error Message	Reason	Countermeasure
Service 0x2A is enabled, but no periodic messages have been configured for Dcm. Please, refer to the Dcm TechRef for SID 0x2A configuration aspect.	You have activated service *ReadDataByPeriodicIdentifier (0x2A)* but have no periodic connection specified.	- Remove the service from the DCM configuration. - Check if any available periodic messages in the communication layers used by DCM. - Check for periodic connections not automatically recognized by the configuration tool.
Vendor specific version numbers of Dcm.c and Dcm.h are inconsistent	The Dcm.c and Dcm.h are not from the same delivery.	- Check for correct sources resp. re-update the sources from the delivered package.
Mismatching OEMs between static and generated code	- Using the DCM code intended for another OEM. - Using wrong configuration tool output for this project.	- Check for correct sources resp. re-update the sources from the delivered package. - Check for using correct configuration tool generation output (*Dynamic Files*).
Not supported DCM/MSR PduR version! No PduR BSWMD file in your MSR project?	Unrecognized/unsupported PduR version is specified.	- Check for correct sources resp. re-update the sources from the delivered package.
Missing information for the supported DTC Extended Data Records! See DCM TechRef!	- The DCM could not retrieve any extended data record information from the DEM module or it is a non-MICROSAR Classic DEM. - In a MICROSAR Classic DEM no extended data records are defined. - In a MICROSAR Classic DEM no DTC refers an extended data record.	- Refer to *4.13.3.1 Reporting Stored DTC Environment Data* for information about this configuration. - Correct the MICROSAR Classic DEM configuration. - Remove the corresponding DCM *ReadDTCInformation (0x19)* sub-function since obviously not required when the DEM does not specify any records.
Missing information for the supported DTC Freeze Frame Records! See DCM TechRef!	- The DCM could not retrieve any snapshot data record information from the DEM module or it is a non-MICROSAR Classic DEM.	- Refer to *4.13.3.1 Reporting Stored DTC Environment Data* for information about this configuration. - Correct the MICROSAR Classic DEM configuration.

Error Message	Reason	Countermeasure
	- In a MICROSAR Classic DEM no snapshot records are defined. - In a MICROSAR DEM no DTC refers a snapshot record. - In a MICROSAR DEM all DTC has been specified to have up to zero (0) snapshot records if calculated snapshot records are chosen.	- Remove the corresponding DCM *ReadDTCInformation (0x19)* sub-function since obviously not required when the DEM does not specify any records.
Unknown DEM AR API interface!	Unrecognized/unsupported DEM API version is specified.	Refer to *8.14 How to Select DEM-DCM Interface Version*.
Too many system timers!	Internal error – DCM design limits reached.	Try reducing the maximum number of schedulable DIDs or number of periodic messages per connection (refer to *ReadDataByPeriodicIdentifier (0x2A)*)
DCM configured to handle OBD DID MIDs via DCM configuration, but MID handling is done by DEM.	This message can be issued only if MSR DEM is used together with MSR DCM. Either the MSR DEM has been configured to handle OBD DTRs as per AR 4.2.2, but at the same time, DCM is configured to do this job too or vice-versa.	Refer to the *8.30 How to Switch Between OBD DTR Support by DCM and* DEM for details on OBD DTR handling and the configuration aspects.
DCM configured to handle OBD DID MIDs via DEM configuration, but no MID handling is done by DEM.		
DCM configured to handle OBD MIDs via DCM configuration, but MID handling is done by DEM.		
DCM configured to handle OBD MIDs via DEM configuration, but no MID handling is done by DEM.		
DID ranges are not allowed if any paged DID is configured!	Incompatible features have been activated.	Refer to *8.24.3 Implementation Limitations* for details on using paged DIDs.
Any other message	Internal inconsistency detection.	Contact Vector.

Table 9-1 Compile time error messages

9.2 Code Generation Time Messages

Here are listed only some of the specific error/warning/information messages that may occur during code generation for MSR DCM.

Message ID	Reason	Description
DCM05010	The **control** operation over a **DID** has been defined to have different execution preconditions for the state group “**session**” in multiple variants.	Refer to *8.19.1.2.1 Handling of State Execution Preconditions of Variant Diagnostic Entities* to learn more about multiple variants and execution preconditions variance.
DCM05011	The **read** operation over a **DID** has been defined to have different execution preconditions for the state group “**session**” in multiple variants.	
DCM05012	The **write** operation over a **DID** has been defined to have different execution preconditions for the state group “**session**” in multiple variants.	
DCM05013	An **RID** has been defined to have different execution preconditions for the state group “**session**” in multiple variants.	
DCM05014	A **diagnostic service** has been defined to have different execution preconditions for the state group “**session**” in multiple variants.	
DCM05015	A **diagnostic sub-service** has been defined to have different execution preconditions for the state group “**session**” in multiple variants.	
DCM05020	The **control** operation over a **DID** has been defined to have different execution preconditions for the state group “**security access**” in multiple variants.	
DCM05021	The **read** operation over a **DID** has been defined to have different execution preconditions for the state group “**security access**” in multiple variants.	
DCM05022	The **write** operation over a **DID** has been defined to have different execution preconditions for the state group “**security access**” in multiple variants.	
DCM05023	A **RID** has been defined to have different execution preconditions for the state group “**security access**” in multiple variants.	
DCM05024	A **diagnostic service** has been defined to have different execution preconditions for the state group “**security access**” in multiple variants.	

Message ID	Reason	Description
DCM05025	A **diagnostic sub-service** has been defined to have different execution preconditions for the state group “**security access**” in multiple variants.	

Table 9-2 Code Generation Time Messages

10 Glossary and Abbreviations

10.1 Glossary

Term	Description
DaVinci Configurator 5	Configuration and generation tool for MICROSAR Classic components
CANdelaStudio	Tool for creating and editing a formal vehicle ECU diagnostic specification

Table 10-1 Glossary

10.2 Abbreviations

Abbreviation	Description
ALFID	Address and Length Format Identifier
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
BCD	Binary Coded Decimal
BSW	Basis Software
C/S	Client/Server (Port)
CDD	Complex Device Driver
CEM	Control Enable Mask
CEMR	CEM Record
DCM	Diagnostic Communication Manager
DDM	Diagnostic Data Modifier
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DDID	Dynamic DID
DID	Data Identifier
DTR	Diagnostic Test Result
ECU	Electronic Control Unit
EWT	Event Window Time
FBL	Flash Boot Loader
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MDG	Mode Declaration Group
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MID	Memory Identifier (in memory service context)
MID	Monitor Identifier (in OBD context)
NRC	Negative Response Code

N_TA	Node Target Address
OBD2	On Board Diagnostics 2
OCY	Operation Cycle
PBS	Post Build Selectable (variant handling)
PBL	Post Build Loadable (variant handling)
PDID	Periodic DID
PID	Parameter Identifier
PPort	Provide Port
RID	Routine Identifier
RoE	Response on Event
RPort	Require Port
RTE	Runtime Environment
S/R	Sender/Receiver (Port)
SADR	Security Access Data Record
SNS	Service Not Supported
SNV	Symbolic Name Value
STRT	Service To Respond To
SWC	Software Component
SWS	Software Specification
TID	Test Identifier
VID	Vehicle Information Identifier
VSG	Vehicle System Group

Table 10-2 Abbreviations

11 Contact

Visit our website for more information on

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

www.vector.com