MICROSAR DCM

Technical Reference

Ford

Version 7.1

Authors	Mishel Shishmanyan, Patrick Rieder, Vitalij Krieger, Thomas Dedler, Alexander Ditte, Savas Ates
Status	Released

Document Information

History

Author	Date	Version	Remarks
Thomas Dedler, Mishel Shishmanyan	2012-08-15	1.00.00	Initial version
Mishel Shishmanyan	2012-09-21	1.01.00	**Added:** *5.19 ReadDataByPeriodicIdentifier ($2A)* *5.22 InputOutputControlByIdentifier ($2F)* *6.5.2.7 ReturnControlToECU()* *6.5.2.8 ResetToDefault()* *6.5.2.9 FreezeCurrentState()* *6.5.2.10 ShortTermAdjustment()* *9.8 How to Jump into the FBL from Service DiagnosticSessionControl ($10)* *9.10 How to Put DCM in a Non-Default Session at ECU Power-On* **Modified:** *Table 6-80 DataServices_<DataName>* *Table 3-4 DET Service IDs*
Mishel Shishmanyan	2012-12-12	1.02.00	**Added:** *5.15 ReadMemoryByAddress ($23)* *5.20 DynamicallyDefineDataIdentifier ($2C)* *5.24 WriteMemoryByAddress ($3D)* *Table 6-43 Dcm_ReadMemory()* *Table 6-44 Dcm_WriteMemory()* **Modified:** *Table 6-50 ConditionCheckRead(),* *Table 6-53 ReadDataLength(),* *Table 6-54 WriteData() (dynamic length),* *Table 6-55 WriteData() (static length),* *Table 6-56 ReturnControlToECU(),* *Table 6-57 ResetToDefault(),* *Table 6-58 FreezeCurrentState(),* *Table 6-59 ShortTermAdjustment()* -"OpStatus"-parameter availability limitation *Table 6-38 <Module>_<DiagnosticService>()* *Table 6-39 <Module>_<DiagnosticService>_<SubService>()*
Mishel Shishmanyan	2013-04-17	1.03.00	No changes
Mishel Shishmanyan	2013-06-28	1.04.00	**Added:**

			Chapters for OBD service 0x01- 0x0A. *6.6.1.2.6 DtrServices* *6.6.1.2.7 RequestControlServices_<TIDName>* *6.6.1.2.8 InfotypeServices_<VEHINFODATA>* *9.11 How to Support Calibrateable Configuration Parameters* **Modified:** *Table 3-2 Not supported AUTOSAR standard conform features* – Removed not supported OBD. *Table 8-3 Limitations to AUTOSAR* – Removed not supported OBD.
Mishel Shishmanyan	2013-08-20	1.05.00	**Added:** *6.6.1.2.9 CallbackDCMRequestServices_<SWC>* *9.12 How and When to Configure Multiple Protocols* *8.1 Deviations* – Added deviation to *CallbackDCMRequestServices_<SWC>* service port. *Table 8-3 Limitations to AUTOSAR* – Added maximum number of supported protocols. – Added maximum number of concurrent client diagnostic connections. **Modified:** *5.14 ReadDataByIdentifier ($22)* *5.22 InputOutputControlByIdentifier ($2F)* – Modified configuration and implementation aspects. *Table 6-85 DtrServices_<MIDName>_<TIDName>* *Table 6-86 RequestControlServices_<TIDName>* – Changed port names according to AR DCM SWS. *Table 6-87 InfotypeServices_<VEHINFODATA>* – Editorial change. *Table 8-3 Limitations to AUTOSAR* – Removed not supported multi-protocol. – Removed not supported multiple buffers.

Thomas Dedler, Mishel Shishmanyan	2013-09-17	2.00.00	**Modified:** *8.1 Deviations* – Removed deviations for DID and RID signals. *3.1 Features* – Removed not supported multi-protocol. – Removed not supported multiple buffers. *6.5.2.12 Start(), 6.5.2.13 Stop(), 6.5.2.14 RequestResults()* – Changed function signatures and descriptions *5.22 InputOutputControlByIdentifier ($2F)* – More details on how optional CEM is supported by DCM. *6.5.2.10 ShortTermAdjustment()* – Removed statement that CEM is included in the controlOptionRecord.
Mishel Shishmanyan	2014-01-14	2.01.00	**Added:** *9.13 How to Select DEM-DCM Interface Version* *9.14 How to Support OBD and UDS over a Single Client Connection* *9.15 How to Use a User Configuration File* *9.16 How to Know When the Security Access Level Changes* **Modified:** *3.4.1 Split Task Functions* – Added configuration aspects. *Table 4-3 Compiler abstraction and memory mapping* – Added calibration parameter memory sections. *5.11 EcuReset ($11)* – Added clarification for request rejection while waiting for reset execution. *5.13 ReadDiagnosticInformation ($19)* – Added support of new sub-functions 0x17-0x19. *5.19 ReadDataByPeriodicIdentifier ($2A)* – Added feature stop periodic reading on changed state. *5.20 DynamicallyDefineDataIdentifier ($2C)* – Added feature clear DDID on changed state.
Mishel Shishmanyan,	2014-04-14	2.02.00	**Added:**

Patrick Rieder			*9.17 How to Deal with the PduR AR version* **Modified:** *Figure 2-2 Interfaces to adjacent modules of the DCM* – NvM added to figure *3.4.1 Split Task Functions* – Reworked chapter – Added support by configuration tool *5.14.4 Configuration Aspects* – Added information about NvRam signal configuration *5.21.4 Configuration Aspects* – Added information about NvRam signal configuration *6.3 Services used by DCM* – Added NvM services used by the DCM *6.4.3.2.1 Dcm_StartOfReception(), 6.4.3.2.3 Dcm_TpRxIndication(), 6.4.3.2.5 Dcm_TpTxConfirmation()* – New version of the APIs for AR 4.1.2 PduR added *8.3 Limitations* – Shared signals between DIDs not supported
Mishel Shishmanyan	2014-10-08	3.00.00	**Added:** *5.16 ReadScalingDataByIdentifier ($24)* *6.5.2.11 GetScalingInformation()* *6.6.2 Managed Mode Declaration Groups* *9.18 Post-build Support* *10 Troubleshooting* **Modified:** *4.1.2 Dynamic Files* – Added post-build data files – Added SWC template file *Table 4-3 Compiler abstraction and memory mapping* – Added post-build data memory mapping *Figure 4-1 Include structure* – Added post-build configuration and other BSW headers files *5.11 EcuReset ($11)* – Sub-functions are now allowed to be user defined. *5.13 ReadDiagnosticInformation ($19)* – Added new implementation aspect. *6.6.1.2.1 DataServices_<DataName>*

			– Added new port operation *GetScalingInformation()* *8.3 Limitations* – Added limitation for support of DIDranges *6.2.1.1 Dcm_Init()* *7.1 Configuration Variants* – Added new post-build variants *9.16 How to Know When the Security Access Level Changes* – Added link to the corresponding mode declaration group.
Mishel Shishmanyan, Vitalij Krieger, Thomas Dedler	2015-01-13	3.01.00	**Added:** *6.5.2.24 IsDidAvailable()* *6.5.2.25 ReadDidData()* *6.5.2.26 WriteDidData()* *9.19 Handling with DID Ranges* *9.20 How to Support DID 0xF186* **Modified:** *5.14.4, 5.23* – Removed WWH-OBD only DIDs/RIDs from examples. *6.3 Services used by DCM* – AR3 support added *6.4.2 ComM, 6.4.3 PduR* – AR3 support added *8.3 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 8-2 Additions/ Extensions to AUTOSAR* – Added AR 3.x integration
Vitalij Krieger, Mishel Shishmanyan	2015-02-06	4.00.00	**Added:** *6.2.1.6 Dcm_InitMemory()* *6.2.3.1 Dcm_GetTesterSourceAddress()* *6.4.4 CanTp* *6.5.1.8<Diagnostic Session Change Notification Callback>* *6.5.1.9<Security Access Change Notification Callback>* *9.21 How to Suppress Responses to Functional Addressed Requests* *9.22 How to Support Interruption on Requests with Foreign N_TA*

			9.23 How to Know When the Diagnostic Session Changes **Modified:** Minor editorial changes *Table 4-3 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 *6.2.1.5 Dcm_GetVersionInfo()* – Specified the digit format of the module's version information. *Figure 4-1 Include structure* – New include structure *4.1.1 Static Files* – New files introduced *4.2 Include Structure* – New include structure *5.17.4 Configuration Aspects* – Added support for single/multiple instace attempt counter, delay timer. *9.16 How to Know When the Security Access Level Changes* – Added new notification type
Alexander Ditte, Mishel Shishmanyan	2015-05-04	4.01.00	**Added:** *6.2.2.5 Dcm_GetSecurityLevelFixedBytes()* *6.4.3.1.1 Dcm_TriggerTransmit()* *6.4.3.2.6 Dcm_TxConfirmation()* *6.5.2.27 GetSecurityAttemptCounter()* *6.5.2.28 SetSecurityAttemptCounter()* *9.24 How to Save RAM using Paged-Buffer for Large DIDs* *9.25 How to Get Security-Access Level Specific Fixed Byte* Values *9.26 How to Extend the Diag Keep Alive Time during Diagnostics* *9.27 How to Recover DCM State Context on ECU Reset/ Power On* **Modified:** Global minor editorial changes *Table 5-26 Service $19: Supported subservices* – Added sub-function 0x42 *5.10.4 Configuration Aspects* – Added session specific timings aspect *5.11 EcuReset ($11)*

			– Added note for resetting to default session if needed. *5.17 SecurityAccess ($27)* – Additions for the new features listed related to this service *5.18.4 Configuration Aspects;* *5.13.4 Configuration Aspects* – Added a hint for external sub-function implementation. *5.22.4 Configuration Aspects* – Removed limitation to static length IO DID for service 0x2F. *5.26.4 Configuration Aspects* – Added missing related configuration parameters *Table 6-79 DCMServices* – Added new operation *Table 3-4 DET Service IDs* – Completed API list – Names converted to hyperlinks for convenience
Mishel Shishmanyan	2015-11-12	5.00.00	**Added:** *6.2.3.2 Dcm_ProcessVirtualRequest()* *6.2.3.3 Dcm_SetSecurityLevel()* *6.2.2.6 Dcm_SetActiveDiagnostic()* *6.5.1.11 Dcm_FilterDidLookUpResult* *6.5.1.12 Dcm_FilterRidLookUpResult* *9.28 How to Define a Diagnostic Connection without USDT* Responses *9.29 How to Handle Multiple Diagnostic Service Variants* **Modified:** Minor editorial changes *5 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. *5.17 SecurityAccess ($27)* – Added external service implementation aspect. *9.11.1 OBD Calibration* – Added information about feature configuration. *5.22* InputOutputControlByIdentifier ($2F) – More events monitored for automatic IO DID

			resetting. *Table 3-4 DET Service IDs* – Added missing DET SIDs *6.5.2.24 IsDidAvailable()* – Removed limitation of synchronous usage. *Table 4-3 Compiler abstraction and memory mapping* – Added DCM_VAR_INIT memory section for 32bit data.
Mishel Shishmanyan	2016-03-01	5.01.00	**Added:** *6.2.2.7 Dcm_GetRequestKind()* **Modified:** Minor editorial changes. *Table 9-10* – Added details for "RID operation". *Table 6-79 DCMServices* – Corrected supported return error codes – Added new operations *5.22 InputOutputControlByIdentifier ($2F)* *6.5.2.7 ReturnControlToECU()* *6.5.2.8 ResetToDefault()* *6.5.2.9 FreezeCurrentState()* *6.5.2.10 ShortTermAdjustment()* – Reworked for support of bitmapped IO DIDs. *Table 3-4 DET Service IDs* – Added new services *8.2 Additions/ Extensions* – Added multi byte external CEMR handling *8.3 Limitations* – Removed limitation for API *IsDidAvailable()*
Mishel Shishmanyan	2016-04-29	5.02.00	 **Modified:** Minor editorial changes.
Mishel Shishmanyan	2016-07-05	7.00.00	**Added:** *9.30 How to Switch Between OBD DTR Support by DCM and DEM* *9.31 How to Enable Support of OBD VIDs with Dynamic Length* *10.2 Code Generation Time Messages* **Modified:** Minor editorial changes. *5.8.4 Configuration Aspects* – Reordered FAQ and added variable data size specific hints.

			Table 6-20 Services used by the DCM – Updated used DEM API. *5.14.4 Configuration Aspects* – *Added configuration aspects for OBD MID DIDs.* *5.18.4 Configuration Aspects* – Added FAQ for “AllNetworks” parameter. *6.5.2.22 GetInfotypeValueData()* – Added AR4.2.2 compatibility. *9.18.1.2 Diagnostic Services Part* – Enabled PBS for diagnostic service. *Table 10-1 Compile time error messages* – Added new messages. *Table 8-1 Deviations to AUTOSAR* – Removed deviation to AR for suppression of NRC 0x7E and 0x7F, since AR 4.2.2 now does require this behavior.
Mishel Shishmanyan	2016-09-22	7.01.00	**Added:** *9.32 How to setup DCM for Sender-Receiver Communication* *9.33 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. *9.1 How to Reduce RAM Usage* – Added information regarding DCM buffer size recommendations integrated in the Configuration 5 tool. *6.5.2.22 GetInfotypeValueData()* – Corrected OpStatus parameter description. *11.2 Abbreviations* – Added “C/S” and “S/R”.

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DiagnosticCommunicationManager.pdf	V4.2.2
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	ISO	ISO 14229-1 UDS	2013
[6]	ISO	ISO 15031-5 OBD	2004
[7]	ISO	ISO 27145-2 WWH-OBD CDD Emissions	2009
[8]	ISO	ISO 27145-3 WWH-OBD CMD	2009
[9]	Vector	TechnicalReference_PostBuildSelectable.pdf	See delivery
[10]	Vector	TechnicalReference_IdentityManager.pdf	See delivery
[11]	Vector	TechnicalReference_Dem.pdf	See delivery

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.

Caution
This symbol calls your attention to warnings.

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.0.0	Initial Version
1.1.0	Added support for diagnostic services: *ReadDataByPeriodicIdentifier ($2A)* *InputOutputControlByIdentifier ($2F)*
1.2.0	Added support for diagnostic service: *DynamicallyDefineDataIdentifier ($2C)* Changed *DataServices_<DataName>* to have either all synchronous or asynchronous operations.
1.3.0	Minor improvements
1.4.0	Support for OBD2 protocol diagnostic services
1.5.0	Support for multi-protocol use cases and protocol prioritization Support resetting of IO control operation at session change/protocol preemption. Support for IO control actual data reporting in the positive response of SID 0x2F. Support for optional *ConditionCheckRead()* DataServices operation
2.0.0	Support for signal interfaces (C/S) for DIDs and RIDs. Extended run time limitation (*How to Reduce DCM Main-Function Run Time Usage*).
2.1.0	Support for DEM AR 4.1.2 API Automatic clear of DDID definition on DCM session/security level change Automatic stop of PDID reading on DCM session/security level change Optional SWC notification on security access level change
2.2.0	Support NvRam signal access for DIDs Support for PduR AR4.1.2 API
3.0.0	Support for diagnostic service *ReadScalingDataByIdentifier ($24)* Support for post-build selectable, loadable, selectable-loadable, deletable for the communication part of DCM.
3.1.0	Support of DID ranges Support for AR3 interfaces (PduR, ComM etc.)
4.0.0	Support of response suppression on functional addressed requests Support of interruption by service request with foreign N_TA New include structure and module refactoring Additional notification on diagnostic session and security access level state transitions.
4.1.0	Support of session specific P2/P2Star timings (*5.10.4 Configuration Aspects*). Non-volatile storage of security access failed attempts (*SecurityAccess*

Component Version	New Features
	($27)). Configurable security level specific fixed bytes to support application seed/key calculation (see *9.25 How to Get Security-Access Level Specific Fixed Byte* Values). Support for paged-buffer reading of DIDs over service *ReadDataByIdentifier ($22)*. Selectable C/S or direct function-calls for service 0x27 application callbacks. Extensible Keep-Alive time period (see *9.26 How to Extend the Diag Keep Alive Time during Diagnostics*) DCM state recovery on reset /power on (*9.27 How to Recover DCM State Context on ECU Reset/ Power*)
5.0.0	Service *InputOutputControlByIdentifier ($2F)* has now improved auto-resetting functionality on any diagnostic state change. Service *TesterPresent ($3E)* can be handled within the application. Refer to its chapter to get information on any limitations. Support of diagnostic connections without response (see *9.28 How to Define a Diagnostic Connection without USDT Responses*) *Support of diagnostic service variant-handling using application help (see 9.29 How to Handle Multiple Diagnostic Service Variants*)
5.1.0	The request status/kind of a DCM diagnostic client can be acquired at any time, using new provided service API *Dcm_GetRequestKind()*. Support for bitmapped IO DIDs with CEMR in service *InputOutputControlByIdentifier ($2F).*
5.2.0	Minor improvements.
7.0.0	Variant handling for the DCM *Diagnostic Services Part.* Improved AR 4.2.2 compatibility regarding: - *How to Switch Between OBD DTR Support by DCM and DEM* - *How to Enable Support of OBD VIDs with Dynamic Length* - API *Dcm_ReadMemory()* resp. *Dcm_WriteMemory()*.
7.1.0	Automatic reporting of RoutineInfoByte parameter in UDS RIDs (see *9.33 How to Support Routine Info Byte with UDS RIDs*) TBD

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4	
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. [4])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

The Autosar DCM is a software component that

- handles the diagnostic communication between the tester and the ECUs application;
- analyzes and interprets the diagnostic communication protocol UDS based on ISO 14229 ([5]);
- implements the handling of all UDS 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;
  - custom ECU mode condition verification (e.g. vehicle speed, etc.)

2.1 How to Read This Document

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

2.1.1 DCM Integration and Basic Operation

We recommend starting with the chapter *4 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.

2.1.2 Diagnostic Service Documentation

Once the DCM is integrated into your project, you will need to know how each diagnostic service, your ECU has to support, is to be configured, implemented and handled by DCM and your application. For learning that, please refer to chapter *5 Diagnostic Service Implementation*.

2.1.3 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 (look into the corresponding *Diagnostic Service Implementation* sub-chapter*)*.

> a special feature it serves (look into the corresponding *Using the DCM* “how to…” sub-chapter)

2.1.4 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 Configurator 5 GCE for faster location of the concrete parameter. Once you have followed such a link in the 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 in order to get a fully functioning configuration.

2.2 Architecture Overview

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

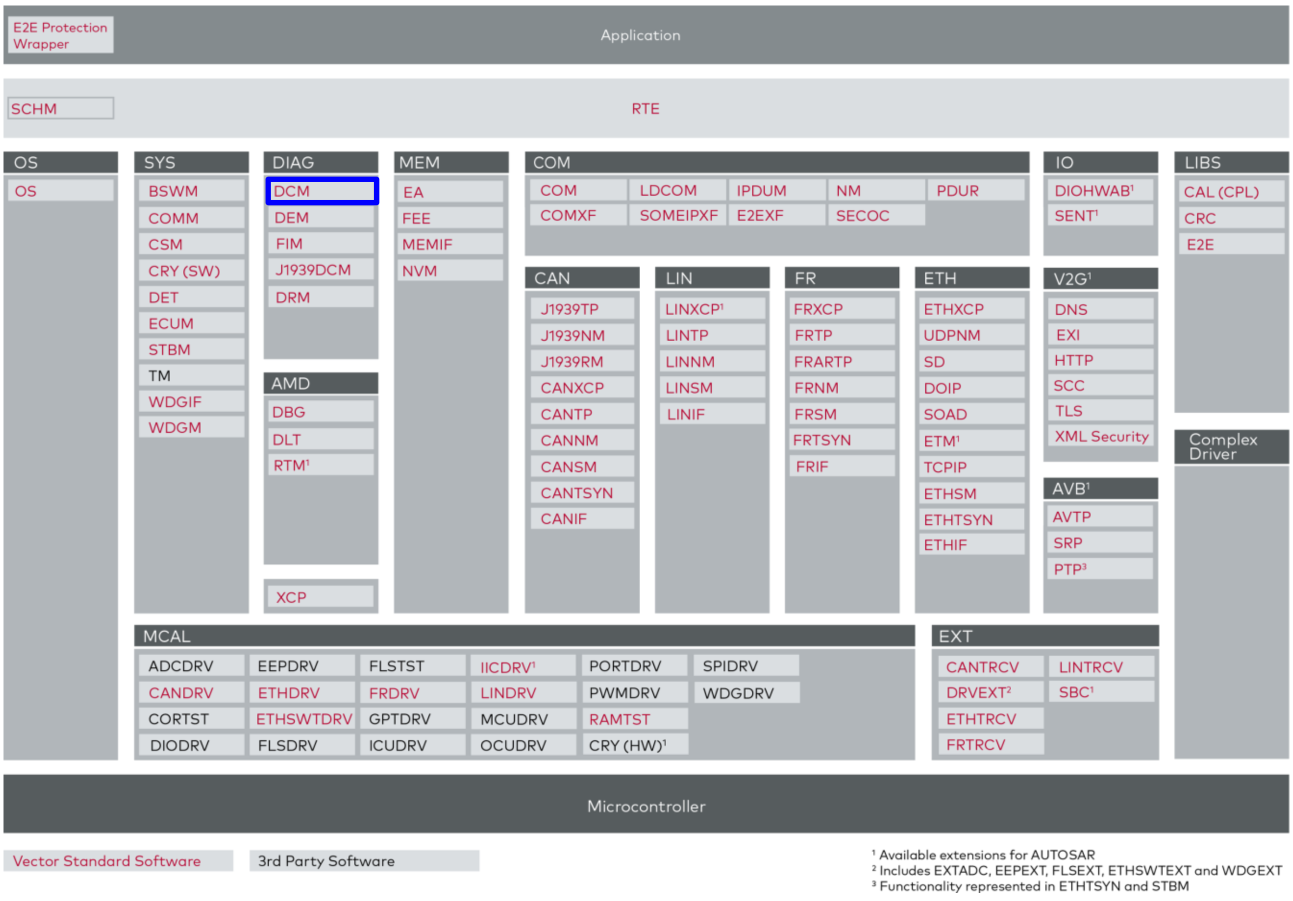

Figure 2-1 AUTOSAR 4.2 Architecture Overview

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

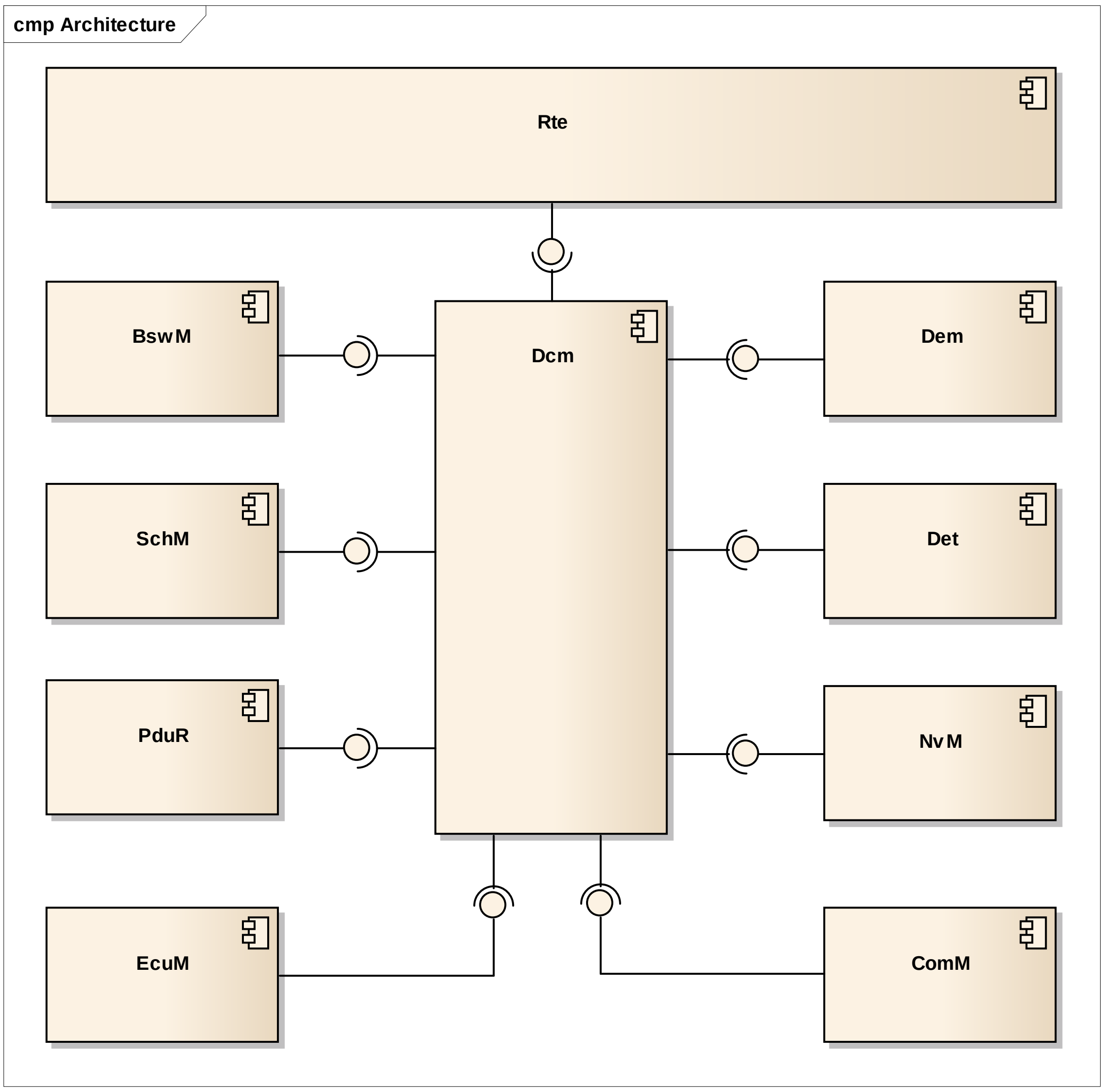

Figure 2-2 Interfaces to adjacent modules of the DCM

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 6.5.2.1 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.

3 Functional Description

3.1 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 3-1 Supported AUTOSAR standard conform features*
- *Table 3-2 Not supported AUTOSAR standard conform features*

For further information of not supported features see also chapter 8.

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

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

The following features specified in *[1]* are supported:

Supported AUTOSAR Standard Conform Features
MICROSAR Identity Manager using Post-Build Selectable All features not listed in *Table 3-2 Not supported AUTOSAR standard conform features* are to be considered as supported.

Table 3-1 Supported AUTOSAR standard conform features

The following features specified in *[1]* are not supported:

Not Supported AUTOSAR Standard Conform Features
No link time configuration support. No post-build support on diagnostic services (only communication). Though an alternative solution is provided (see *9.29*)

Table 3-2 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*
Run time optimized DCM DEM interface for low CPU load.
Native AR 4.0.3 and AR 4.1.2 DEM API version support.
Support for sub-functions 0x17, 0x18 and 0x19 of service *ReadDiagnosticInformation ($19)* according to *[5]*.

Features Provided Beyond The AUTOSAR Standard
Optional notification on security access level change
Extensible keep-alive time period: *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*

Table 3-3 Features provided beyond the AUTOSAR standard

3.2 Initialization

At ECU power-on boot (or any reset situation) DCM must be initialized by calling the API *Dcm_Init()*.

3.3 States

DCM manages currently the following state machines:

- Diagnostic session states (managed by service *DiagnosticSessionControl ($10)*)
- Security access states (managed by service *SecurityAccess ($27)*)
- ECU Communication activity (managed by the ComM)
- DTC setting allowance (managed by the Dem)

3.4 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

3.4.1 Split Task Functions

3.4.1.1 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.

3.4.1.2 Configuration Aspects

Per default DCM has only one *Dcm_MainFunction()* i.e. has no split tasks as specified in *[1]*. In order to enable split task usage in DCM, you have to 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

3.4.1.3 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()*).

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], 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 6.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	*Dcm_StartOfReception()*
0x01	*Dcm_Init()*
0x02	*Dcm_CopyRxData()* (AR4) / *Dcm_ProvideRxBuffer()* (AR3)
0x03	*Dcm_TpRxIndication()* (AR4) / *Dcm_RxIndication()* (AR3)
0x04	*Dcm_CopyTxData()* (AR4) / *Dcm_ProvideTxBuffer()* (AR3)
0x05	*Dcm_TpTxConfirmation()* (AR4) / *Dcm_TxConfirmation()* (AR3)
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()*
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()*
0xA0	**Depricated from DCM 5.00.00 and mapped to "*DCM internal function*".** DCM internal diagnostic service processor
0xA1	*Dcm_TxConfirmation()*
0xA2	*Dcm_TriggerTransmit()*
0xA3	*Dcm_ProvideRecoveryStates()*
0xA4	*Dcm_OnRequestDetection()*

Service ID	Service
0xA6	*Dcm_GetTesterSourceAddress()*
0xA7	*Dcm_GetSecurityLevelFixedBytes()*
0xA8	*Dcm_ProcessVirtualRequest()*
0xA9	*Dcm_SetSecurityLevel()*
0xAA	*ReadData() (paged-data-reading)*
0xAB	*Dcm_GetRequestKind()*
0xF0	DCM internal function

Table 3-4 DET Service IDs

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

Error Code		Description
0x01	DCM_E_INTERFACE_TIMEOUT	Timeout during interaction with other module
0x02	DCM_E_INTERFACE_RETURN_VALUE	Return value of called API is out of range
0x03	DCM_E_INTERFACE_BUFFER_OVERFLOW	Boundary check of provided buffers fails
0x05	DCM_E_UNINIT	Executing program code before the DCM is initialized
0x06	DCM_E_PARAM	API call with invalid parameter value
0x07	DCM_E_PARAM_POINTER	API call with invalid/null pointer parameter
0x40	DCM_E_ILLEGAL_STATE	Internal DCM error, reaching an unexpected state
0x41	DCM_E_INVALID_CONFIG	Inconsistent configuration

Table 3-5 Errors reported to DET

3.5.2 Production Code Error Reporting

Production code related errors are not supported by DCM.

4 Integration

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

4.1 Scope of Delivery

The delivery of the DCM contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 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_Ext.c	■		This is the implementation source file of the DCM with Autosar extensions (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 has to be included by the application if an interaction with DCM is needed.**
Dcm_Int.h	■		This is the header file containing internal APIs of DCM between the core- and extension parts. **This file must not be included by any other source file except of the DCM own ones.**
Dcm_Cbk.h	■		This file contains all function prototypes of APIs called by other BSW-C (i.e. Pdu-R, ComM, etc.).
Dcm_Types.h	■		This file contains all data types that shall be visible to the other components interacting with DCM.
Dcm_Core.h Dcm_CoreInt.h Dcm_CoreCbk.h Dcm_CoreTypes.h	■		All these files belong to the DCM core part. **None of these files must be included by an external source code.**
Dcm_Ext.h Dcm_ExtInt.h Dcm_ExtCbk.h Dcm_ExtTypes.h	■		All these files belong to the DCM extension part. **None of these files must be included by an external source code.**
Dcm_bswmd.arxml	■		This file contains all DCM configuration parameters' definitions.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the 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.
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 4-2 Generated files

4.2 Include Structure

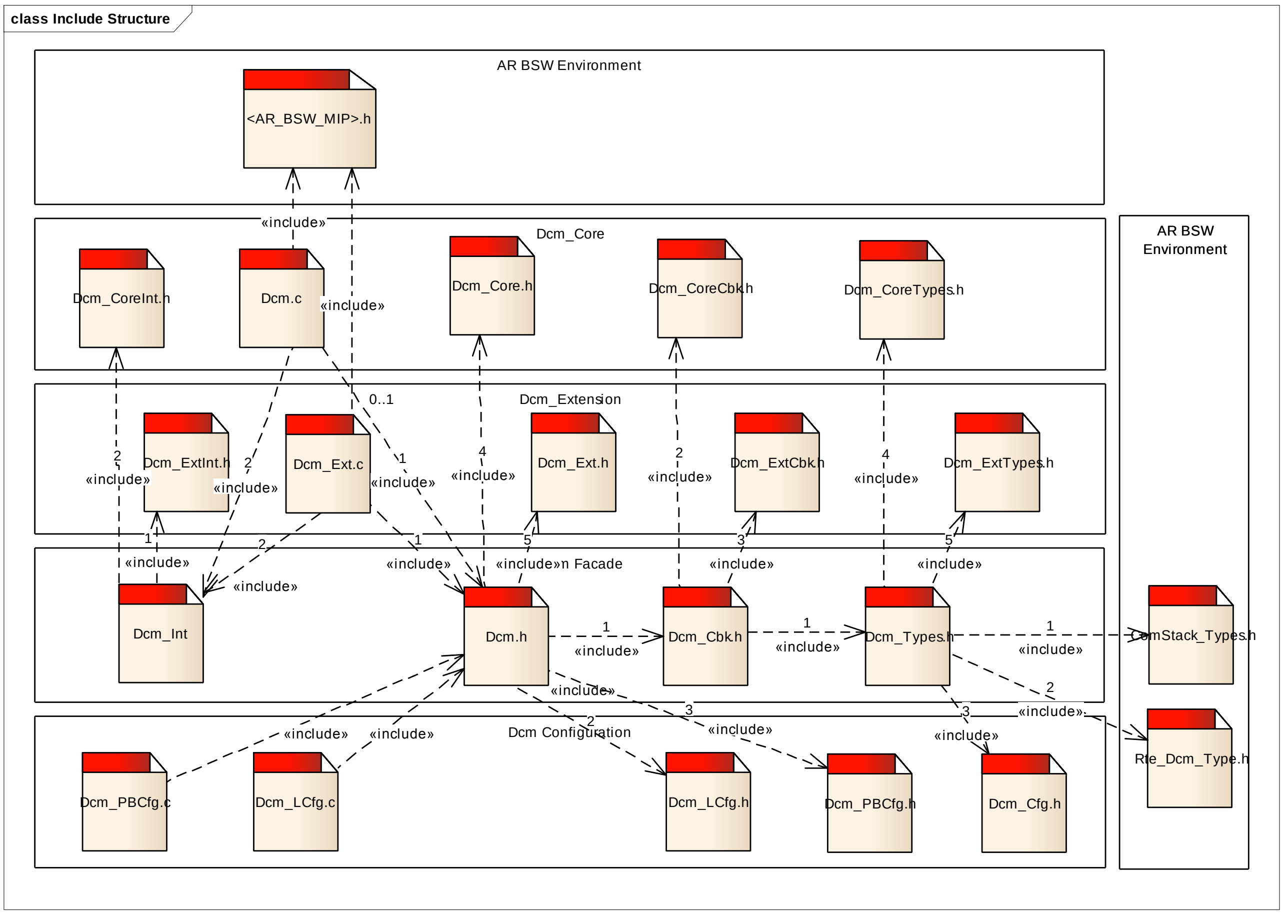

Figure 4-1 Include structure

4.3 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_NOINIT_8 DCM_STOP_SEC_VAR_NOINIT_8				■							
DCM_START_SEC_VAR_NOINIT_16 DCM_STOP_SEC_VAR_NOINIT_16				■							
DCM_START_SEC_VAR_INIT_32 DCM_STOP_SEC_VAR_INIT_32					■						
DCM_START_SEC_VAR_NOINIT_32 DCM_STOP_SEC_VAR_NOINIT_32				■							
DCM_START_SEC_VAR_NOINIT_UNSPECIFIED DCM_STOP_SEC_VAR_NOINIT_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 4-3 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).

4.4 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 have to 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.

4.5 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 will have to store it into temporary RAM location in your application software, before starting the write process.

5 Diagnostic Service Implementation

The main goal of the DCM is to handle the diagnostic protocol services, defined by [5]. 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.

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 Configurator 5 and handle it within your application. If you try to specify an invalid service identifier, the 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 actually the application implementation of these services still can handle any by itself.

5.1 RequestCurrentPowertrainDiagnosticData ($01)

5.1.1 Functionality

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

5.1.2 Required Interfaces

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

5.1.3 Implementation Aspects

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

Table 5-1 Service $01: Implementation types

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

Table 5-2 Service $01: Supported subservices

This service is fully implemented by DCM.

5.1.4 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 has to 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 ($22)* DIDs 0xF400 – 0xF4FF), the corresponding DIDs, **including the “availability DIDs”** shall be explicitly defined within the DCM configuration. This is required in order to support the optional read access condition checks on a DID operation.

Refer to *ReadDataByIdentifier ($22)* 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

5.2 RequestPowertrainFreezeFrameData ($02)

5.2.1 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.

5.2.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.2.3 Implementation Aspects

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

Table 5-3 Service $02: Implementation types

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

Table 5-4 Service $02: Supported subservices

This service is fully implemented by DCM.

5.2.4 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 has to 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.

5.3 RequestEmissionRelatedDTC ($03)

5.3.1 Functionality

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

5.3.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.3.3 Implementation Aspects

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

Table 5-5 Service $03: Implementation types

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

Table 5-6 Service $03: Supported subservices

This service is fully implemented by DCM.

5.3.4 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;

5.4 ClearEmissionRelatedDTC ($04)

5.4.1 Functionality

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

5.4.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.4.3 Implementation Aspects

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

Table 5-7 Service $04: Implementation types

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

Table 5-8 Service $04: Supported subservices

This service is fully implemented by DCM.

5.4.4 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;

5.5 RequestOnBoardMonitorTestResults ($06)

5.5.1 Functionality

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

5.5.2 Required Interfaces

- If service handled by DCM:
  - *DtrServices* (if no OBD DTR support by DEM)
  - Refer to chapter *6.3 Services used by DCM* for the DEM component (if OBD DTR is supported by DEM).
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.5.3 Implementation Aspects

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

Table 5-9 Service $06: Implementation types

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

Table 5-10 Service $06: 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:
- *5.5.4 Configuration Aspects*
- *9.30 How to Switch Between OBD DTR Support by DCM and DEM*

5.5.4 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 ($22)* DIDs 0xF600 – 0xF6FF), the corresponding DIDs, **including the “availability DIDs”** shall be explicitly defined within the DCM configuration. This is required in order to support the optional read access condition checks on a DID operation.

Refer to *ReadDataByIdentifier ($22)* for more details about OBD DID configuration particularities.

5.6 RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle ($07)

5.6.1 Functionality

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

5.6.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.6.3 Implementation Aspects

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

Table 5-11 Service $07: Implementation types

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

Table 5-12 Service $07: Supported subservices

This service is fully implemented by DCM.

5.6.4 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;

5.7 RequestControlOfOnBoardSystemTestOrComponent ($08)

5.7.1 Functionality

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

5.7.2 Required Interfaces

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

5.7.3 Implementation Aspects

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

Table 5-13 Service $08: Implementation types

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

Table 5-14 Service $08: Supported subservices

This service is fully implemented by DCM.

5.7.4 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/DcmDspRequestControlInBuff erSize

> 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 ($31)* DIDs 0xE000 – 0xE0FF), the corresponding RIDs, **including the “availability RIDs”** shall be explicitly defined within the DCM configuration. This is required in order to support the optional control access condition checks on a RID.

Refer to *RoutineControl ($31)* for more details about OBD RID configuration particularities.

5.8 RequestVehicleInformation ($09)

5.8.1 Functionality

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

5.8.2 Required Interfaces

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

5.8.3 Implementation Aspects

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

Table 5-15 Service $09: Implementation types

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

Table 5-16 Service $09: Supported subservices

This service is fully implemented by DCM.

5.8.4 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 MIDs shall be also readable by the corresponding UDS service (i.e. *ReadDataByIdentifier ($22)* DIDs 0xF800 – 0xF8FF), the corresponding DIDs, **including the “availability DIDs”** shall be explicitly defined within the DCM configuration. This is required in order to support the optional read access condition checks on a DID operation.

Refer to *ReadDataByIdentifier ($22)* 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

5.9 RequestEmissionRelatedDTCsWithPermanentStatus ($0A)

5.9.1 Functionality

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

5.9.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.9.3 Implementation Aspects

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

Table 5-17 Service $0A: Implementation types

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

Table 5-18 Service $0A: Supported subservices

This service is fully implemented by DCM.

5.9.4 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;

5.10 DiagnosticSessionControl ($10)

5.10.1 Functionality

This service manages the diagnostic session state in the ECU.

5.10.2 Required Interfaces

> *DcmDiagnosticSessionControl*

5.10.3 Implementation Aspects

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

Table 5-19 Service $10: Implementation types

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

Table 5-20 Service $10: Supported subservices

This service is fully implemented by DCM.

5.10.4 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 as long as 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 to 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

5.11 EcuReset ($11)

5.11.1 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. The communication reaction (reject or ignore new request) is dependent on the DCM configuration (see below).

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.

5.11.2 Required Interfaces

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

5.11.3 Implementation Aspects

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

Table 5-21 Service $11: Implementation types

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

Table 5-22 Service $11: Supported subservices

All in *Table 5-22 Service $11: 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.

5.11.4 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

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

> If one of the following sub-functions: 0x01-0x03 is to be supported, the DCM will either reject by NRC 0x21 or ignore any request received while waiting for the reset execution accomplishment. The concrete reaction depends on the setting:
/Dcm/DcmConfigSet/DcmDsl/DcmDslDiagResp/DcmDslDiagRespOnSecondDeclinedRequest (refer also to *9.4 How to Handle Multiple Diagnostic Clients Simultaneously*).

5.12 ClearDiagnosticInformation ($14)

5.12.1 Functionality

This service clears the stored fault memory content.

5.12.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.12.3 Implementation Aspects

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

Table 5-23 Service $14: Implementation types

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

Table 5-24 Service $14: Supported subservices

This service is fully implemented by DCM.

5.12.4 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.

5.13 ReadDiagnosticInformation ($19)

5.13.1 Functionality

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

5.13.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.13.3 Implementation Aspects

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

Table 5-25 Service $19: Implementation types

Implementation / Subservice ID	internal only	internal or external	external only	not allowed
0x00				■
0x01 … 0x15		■		
0x16			■	
0x17 … 0x19		■		
0x1A … 0x41			■	
0x42		■		
0x43 … 0x7E			■	
0x7F … 0xFF				■

Table 5-26 Service $19: Supported subservices

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

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

5.13.3.1 Reporting Stored DTC Environment Data

For all snapshot and extended data record sub-functions, DCM module requires additional input from the ECU configuration. In order 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.

Note
If you use this DCM module together with the MICROSAR DEM, it is not necessary to use or change this configuration. DCM will automatically take the DEM settings regarding the supported record numbers.

5.13.4 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 5-26 Service $19: 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 *[5]* 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.

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

> If a non-MICROSAR DEM 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

5.14 ReadDataByIdentifier ($22)

5.14.1 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 *5.14.4 Configuration Aspects* for more details).

5.14.2 Required Interfaces

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

5.14.3 Implementation Aspects

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

Table 5-27 Service $22: Implementation types

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

Table 5-28 Service $22: 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 using DID ranges, please read carefully chapter *9.19 Handling with DID Ranges* to learn important particularities.

FAQ
In case very large DID data has to be carried out from the application an optimized data reading process can be used to save RAM. For details please refer to *9.24 How to Save RAM using Paged-Buffer for Large DIDs.*

5.14.4 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 has to 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 *9.30 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 *[6]*.
- 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.

5.15 ReadMemoryByAddress ($23)

5.15.1 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 *5.15.4 Configuration Aspects*.

5.15.2 Required Interfaces

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

5.15.3 Implementation Aspects

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

Table 5-29 Service $23: Implementation types

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

Table 5-30 Service $23: 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 ($2C)*.

5.15.4 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

5.16 ReadScalingDataByIdentifier ($24)

5.16.1 Functionality

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

5.16.2 Required Interfaces

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

5.16.3 Implementation Aspects

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

Table 5-31 Service $24: Implementation types

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

Table 5-32 Service $24: 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 $24 under the corresponding parameter (e.g. DcmDsdSidTabSecurityLevelRef) in /Dcm/DcmConfigSet/DcmDsd/DcmDsdServiceTable/DcmDsdService

5.16.4 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

5.17 SecurityAccess ($27)

5.17.1 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.

5.17.2 Required Interfaces

The following interfaces must be available when service $27 is used:

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

5.17.3 Implementation Aspects

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

Table 5-33 Service $27: Implementation types

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

Table 5-34 Service $27: Supported subservices

By default this service is fully implemented by DCM. If the internal implementation is used, the following specifics have to 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.

> 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 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 *9.25 How to Get Security-Access Level Specific Fixed Byte* Values.

5.17.4 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 *9.16 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

5.18 CommunicationControl ($28)

5.18.1 Functionality

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

5.18.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the BswM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.18.3 Implementation Aspects

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

Table 5-35 Service $28: 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 5-36 Service $28: 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.

5.18.4 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 5-36 Service $28: 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 *[5]* 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

5.19 ReadDataByPeriodicIdentifier ($2A)

5.19.1 Functionality

This service provides read access to data structures within the ECU, marked by a periodic identifier (PDID). These are all DIDs in range [$F200 – $F2FF].

The tester may schedule multiple PDIDs in a single request. The maximum allowed PDID list length is configurable (refer to *5.19.4 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 changes. Refer to *5.19.4 Configuration Aspects* for details about this setting.

FAQ
Only periodic responses of type 2 (UUDT) are supported, as the latest versions of [5] require.

5.19.2 Required Interfaces

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

5.19.3 Implementation Aspects

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

Table 5-37 Service $2A: Implementation types

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

Table 5-38 Service $2A: 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.

Caution
If you intend using DID ranges, please read carefully chapter *9.19 Handling with DID Ranges* to learn important particularities.

5.19.4 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

> 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 [$F200-$F2FF].

> 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 clients 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 clients 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

5.20 DynamicallyDefineDataIdentifier ($2C)

5.20.1 Functionality

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

Optionally, DCM is able to clear automatically any already defined DDID that cannot be accessed any more, after a diagnostic session/security access level changes. This also implies that a periodic DDID will be also removed from the periodic scheduler (*ReadDataByPeriodicIdentifier ($2A)*). Refer to *5.20.4 Configuration Aspects* for details about this setting.

5.20.2 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>()*

5.20.3 Implementation Aspects

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

Table 5-39 Service $2C: Implementation types

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

Table 5-40 Service $2C: Supported subservices

This service is fully implemented by DCM corresponding to the *[5]*.

Caution
If you intend using DID ranges, please read carefully chapter *9.19 Handling with DID Ranges* to learn important particularities.

5.20.4 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 verifies always the session, security and mode-rule preconditions of a DDID when it is requested by a diagnostic client. You can configure DCM additionally to check also 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 clear automatically 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 ($22)*) or periodically (*ReadDataByPeriodicIdentifier ($2A)*) 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.

5.21 WriteDataByIdentifier ($2E)

5.21.1 Functionality

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

5.21.2 Required Interfaces

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

5.21.3 Implementation Aspects

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

Table 5-41 Service $2E: Implementation types

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

Table 5-42 Service $2E: 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 using DID ranges, please read carefully chapter *9.19 Handling with DID Ranges* to learn important particularities.

5.21.4 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 has to be referenced: /Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataBlockIdRef

5.22 InputOutputControlByIdentifier ($2F)

5.22.1 Functionality

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

5.22.2 Required Interfaces

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

5.22.3 Implementation Aspects

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

Table 5-43 Service $2F: Implementation types

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

Table 5-44 Service $2F: 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.

IO DID Data Handling in DCM and Application
According to *[5]* 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 is able to 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 details in the next paragraphs.

Packeted IO DID with all signals which have 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 *[5]*. Then at run time 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 *5.22.4 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.

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 *5.22.4 Configuration Aspects* and the detailed parameter description in the configuration tool.

5.22.4 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

> Which IO operation is supported by the IO DID is defined within the container:
/Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidAccess/DcmDspDidControl
. There you have to 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/DcmDspDidIoEnableMaskSize.

FAQ

Particularities of an IO DID configuration:

- If the IO DID has read operation (i.e. accessible via *ReadDataByIdentifier ($22)*) 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.
- An IO DID with read operation shall never has "*ConditionCheckRead()"* operation. For details, please refer to *5.14.4 Configuration Aspects* of service *ReadDataByIdentifier ($22)*. 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 has to 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 one byte stream), then you have to 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, in order 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>`.

5.23 RoutineControl ($31)

5.23.1 Functionality

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

5.23.2 Required Interfaces

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

5.23.3 Implementation Aspects

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

Table 5-45 Service $31: Implementation types

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

Table 5-46 Service $31: 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 back from the corresponding service port, implemented by the application.

5.23.4 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

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 *[6]*.
- 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 *[6]* 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.

5.24 WriteMemoryByAddress ($3D)

5.24.1 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 *5.15.4 Configuration Aspects*.

5.24.2 Required Interfaces

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

5.24.3 Implementation Aspects

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

Table 5-47 Service $3D: Implementation types

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

Table 5-48 Service $3D: 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.

5.24.4 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

5.25 TesterPresent ($3E)

5.25.1 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.)

5.25.2 Required Interfaces

- If service handled by DCM:
  - No interfaces required for this services.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.25.3 Implementation Aspects

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

Table 5-49 Service $3E: Implementation types

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

Table 5-50 Service $3E: 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.

5.25.4 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

> 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 and cannot be overridden by an application implementation.

5.26 ControlDTCSetting ($85)

5.26.1 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).

5.26.2 Required Interfaces

- If service handled by DCM:
  - Refer to chapter *6.3 Services used by DCM* for the DEM component.
- If service handled by the application:
  - *<Module>_<DiagnosticService>()*

5.26.3 Implementation Aspects

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

Table 5-51 Service $85: Implementation types

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

Table 5-52 Service $85: Supported subservices

This service is completely implemented by DCM.

5.26.4 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

- 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 accept also 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

6 API Description

For an interfaces overview please see Figure 2-2.

6.1 Type Definitions

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

6.1.1 Dcm_ProtocolType

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

Table 6-1 Dcm_ProtocolType

6.1.2 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 ($28)* 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 has to exist in order 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 ($85)* 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:	`FALSE` – DCM will not call the ControlDTCSetting DEM API.
			`TRUE` –

Struct Element Name	C-Type	Description	Value Range
		ControlDTCSetting ($85) is enabled in DCM	DCM will perform a ControlDTCSetting operation for "disabling DTC" as for an external diagnostic request for *ControlDTCSetting ($85)*.
SessionLevel	uint8 (typically)	New diagnostic session. Note: This is not the session level as defined by AR. It is 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 DCM internal value. Exist-Condition: If *SecurityAccess ($27)* 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.
Signature	uint32	Magic number for data consistency check between stored and to be recovered data block.	A configuration dependent value.

Table 6-2 Dcm_RecoveryInfoType

6.2 Services provided by DCM

6.2.1 Administrative

6.2.1.1 Dcm_Init()

Prototype	
`void Dcm_Init ( 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 *9.18.2.1 Error Detection and Handling*	
Particularities and Limitations	
> ServiceID = 0x01 > This function is not reentrant. > This function is synchronous.	

Table 6-3 Dcm_Init()

6.2.1.2 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 = 37 > This function is not reentrant. > This function is synchronous.

Table 6-4 Dcm_MainFunction()

6.2.1.3 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 = 37 > This function is not reentrant. > This function is synchronous.	

Table 6-5 Dcm_MainFunctionTimer()

6.2.1.4 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 = 37 > This function is not reentrant. > This function is synchronous.	

Table 6-6 Dcm_MainFunctionWorker()

6.2.1.5 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. **Note:** **Starting with DCM 4.00.00, the version information is decimal coded.**	
Particularities and Limitations	
> ServiceID = 0x24 > This function is reentrant. > This function is synchronous.	

Table 6-7 Dcm_GetVersionInfo()

6.2.1.6 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_* (refer to *4.3 Compiler Abstraction and Memory Mapping*) sections 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 6-8 Dcm_InitMemory()

6.2.1.7 Dcm_ProvideRecoveryStates()

Prototype	
`Std_ReturnType Dcm_ProvideRecoveryStates (`*`Dcm_RecoveryInfoType*`*` RecoveryInfo)`	
Parameter	
`RecoveryInfo`	Contains all the information that has to 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 *9.27 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 6-9 Dcm_ProvideRecoveryStates()

6.2.2 SWC

6.2.2.1 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.	

Particularities and Limitations
> ServiceID = 0x0F > This function is reentrant. > This function is synchronous.

Table 6-10 Dcm_GetActiveProtocol()

6.2.2.2 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.	
Particularities and Limitations	
> ServiceID = 0x0D > This function is reentrant. > This function is synchronous.	

Table 6-11 Dcm_GetSecurityLevel()

6.2.2.3 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 6-12 Dcm_GetSesCtrlType()

6.2.2.4 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 6-13 Dcm_ResetToDefaultSession()

6.2.2.5 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 in order 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 provide fixed bytes information.	

Table 6-14 Dcm_GetSecurityLevelFixedBytes()

6.2.2.6 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 see *[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 6-15 Dcm_SetActiveDiagnostic()

6.2.2.7 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.	

Table 6-16 Dcm_GetRequestKind()

6.2.3 General Purpose

6.2.3.1 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 E_NOT_OK: an error occurred, the `TesterSourceAddress` has no valid value
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.	

Table 6-17 Dcm_GetTesterSourceAddress()

6.2.3.2 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.
`Length`	The length of the diagnostic request located in the `Data` buffer.
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 6-18 Dcm_ProcessVirtualRequest()

6.2.3.3 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 ($27)* 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. In order 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 ($27)* or a *Confirmation()* if configured.	
Particularities and Limitations	
> ServiceID = 0xA9 > This function is not reentrant. > This function is synchronous. > Available only if service *SecurityAccess ($27)* is supported in the ECU but handled within the DCM application.	

Table 6-19 Dcm_SetSecurityLevel()

6.3 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_GetFreezeFrameDataByRecord / 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]GetSizeOfExtendedDataRecordByDTC Dem_[Dcm]GetSizeOfFreezeFrameByDTC Dem_[Dcm]GetTranslationType Dem_[Dcm]GetDTCByOccurrenceTime Dem_[Dcm]DisableDTCSetting Dem_[Dcm]EnableDTCSetting Dem_[Dcm]GetDTCOfOBDFreezeFrame Dem_[Dcm]ReadDataOfOBDFreezeFrame Dem_DcmGetAvailableOBDMIDs Dem_DcmGetNumTIDsOfOBDMID Dem_DcmGetDTRData
BswM	**For AUTOSAR 4.x Environment** BswM_Dcm_CommunicationMode_CurrentState BswM_Dcm_ApplicationUpdated **For AUTOSAR 3.x Environment** BswM_Dcm_RequestCommunicationMode
Det	Det_ReportError
ComM	ComM_DCM_ActiveDiagnostic ComM_DCM_InactiveDiagnostic
PduR	PduR_DcmTransmit

Component	API
SchM	**For AUTOSAR 4.x Environment** SchM_Enter_Dcm_DCM_EXCLUSIVE_AREA_0 SchM_Exit_Dcm_DCM_EXCLUSIVE_AREA_0 **For AUTOSAR 3.x Environment** SchM_Enter_Dcm SchM_Exit_Dcm
NvM	NvM_ReadBlock NvM_WriteBlock NvM_CancelJobs NvM_SetBlockLockStatus NvM_GetErrorStatus NvM_GetDcmBlockId
EcuM	EcuM_BswErrorHook

Table 6-20 Services used by the DCM

6.4 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.

6.4.1 <Module>

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

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

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

6.4.1.1 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 6-21 Dcm_ExternalProcessingDone()

6.4.1.2 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 the 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 6-22 Dcm_ExternalSetNegResponse()

6.4.2 ComM

The DCM supports the ComM interface according to AR3 and AR4. By default, AR4 is used. AR3 is only available in deliveries which are preconfigured accordingly.

6.4.2.1 Dcm_ComM_NoComModeEntered()

Prototype
ComM AR 4.x.x
void **Dcm_ComM_NoComModeEntered** (uint8 NetworkId)
ComM AR 3.x.x
void **Dcm_ComM_NoComModeEntered** (void)

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 6-23 Dcm_ComM_NoComModeEntered()

6.4.2.2 Dcm_ComM_SilentComModeEntered()

Prototype	
ComM AR 4.x.x	
`void` **`Dcm_ComM_SilentComModeEntered`** `( uint8 NetworkId )`	
ComM AR 3.x.x	
`void` **`Dcm_ComM_SilentComModeEntered`** `( void )`	
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 6-24 Dcm_ComM_SilentComModeEntered()

6.4.2.3 Dcm_ComM_FullComModeEntered()

Prototype
ComM AR 4.x.x
`void` **`Dcm_ComM_FullComModeEntered`** `( uint8 NetworkId )`

ComM AR 3.x.x	
`void` **`Dcm_ComM_FullComModeEntered`** `( void )`	
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 6-25 Dcm_ComM_FullComModeEntered()

6.4.3 PduR

The DCM supports different versions of the PduR interface. For details regarding the configuration, please refer to chapter *9.17 How to Deal with the PduR AR version*.

6.4.3.1 All AUTOSAR Versions

6.4.3.1.1 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 6-26 Dcm_TriggerTransmit ()

6.4.3.2 AUTOSAR 4

6.4.3.2.1 Dcm_StartOfReception()

Prototype	
PduR AR 4.0.3 (DCM Version >= 1.00.00)	
`BufReq_ReturnType` **`Dcm_StartOfReception`** `( PduIdType DcmRxPduId, PduLengthType TpSduLength, PduLengthType* RxBufferSizePtr )`	
PduR AR 4.1.2 (DCM Version >= 2.02.00)	
`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. > The prototype of the API depends on the AR version of the PduR (please refer to chapter *9.17 How to Deal with the PduR AR version*).	

Table 6-27 Dcm_StartOfReception()

6.4.3.2.2 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). An 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 an 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 6-28 Dcm_CopyRxData()

6.4.3.2.3 Dcm_TpRxIndication()

Prototype	
PduR AR 4.0.3 (DCM Version >= 1.00.00)	
`void` **`Dcm_TpRxIndication`** `( PduIdType DcmRxPduId, NotifResultType Result )`	
PduR AR 4.1.2 (DCM Version >= 2.02.00)	
`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`	PduR AR 4.0.3: NTFRSLT_OK: The complete N-PDU has been received and is stored in the receive buffer. any other value: The N_PDU has not been received; the receive buffer can be unlocked by the DCM. PduR AR 4.1.2: 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. > The prototype of the API depends on the AR version of the PduR (please refer to chapter *9.17 How to Deal with the PduR AR version*).	

Table 6-29 Dcm_TpRxIndication()

6.4.3.2.4 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 has to 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 have to 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 amount 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 6-30 Dcm_CopyTxData()

6.4.3.2.5 Dcm_TpTxConfirmation()

Prototype	
PduR AR 4.0.3 (DCM Version >= 1.00.00)	
`void` **`Dcm_TpTxConfirmation`** `( PduIdType DcmTxPduId, NotifResultType Result )`	
PduR AR 4.1.2 (DCM Version >= 2.02.00)	
`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`	PduR AR 4.0.3: NTFRSLT_OK if the complete N-PDU has been transmitted. any other value: an error occurred during transmission, the DCM can unlock the transmit buffer. PduR AR 4.1.2: 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. > The prototype of the API depends on the AR version of the PduR (please refer to chapter *9.17 How to Deal with the PduR AR version*).	

Table 6-31 Dcm_TpTxConfirmation()

6.4.3.2.6 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 6-32 Dcm_TxConfirmation()

6.4.3.3 AUTOSAR 3

6.4.3.3.1 Dcm_ProvideRxBuffer()

Prototype	
`BufReq_ReturnType` **`Dcm_ProvideRxBuffer`** `( PduIdType DcmRxPduId, PduLengthType TpSduLength, PduInfoType** PduInfoPtr )`	
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.
`TpSduLength`	The overall length of the message being received.
`PduInfoPtr`	Pointer to pointer to PduInfoType containing data pointer and length of a receive buffer, which is provided by the DCM. Note that certain TP's will put an initial value inside the length buffer, which is the minimal size of the RxBuffer. This value is ignored by the DCM.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: buffer has been successfully provided. BUFREQ_E_OVFL: no buffer provided, available buffer is too small. BUFREQ_E_NOT_OK: no buffer provided, request failed.
Functional Description	
Called at least once upon reception by a lower layer TP to request a buffer, where the received data will be stored. When called multiple times, the DCM expects that the previously provided buffer has been filled up completely.	
Particularities and Limitations	
> ServiceID = 0x02 > Reentrant for different PduIds. Non reentrant for the same PduId. > This function is synchronous.	

Table 6-33 Dcm_ProvideRxBuffer ()

6.4.3.3.2 Dcm_RxIndication()

Prototype	
`void` **`Dcm_RxIndication`** `( PduIdType DcmRxPduId, NotifResultType 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`	NTFRSLT_OK: The complete I-PDU has been received and is stored in the receive buffer. any other value: The I-PDU has not been received; the receive buffer can be unlocked by the DCM.
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 6-34 Dcm_RxIndication()

6.4.3.3.3 Dcm_ProvideTxBuffer()

Prototype	
`BufReq_ReturnType` **`Dcm_ProvideTxBuffer`** `( PduIdType DcmTxPduId, PduInfoType** PduInfoPtr, PduLengthType Length )`	
Parameter	
`DcmTxPduId`	Identifies the DCM data to be sent. This information is used to derive the PCI information within the transport protocol. The value has to be same as in the according service call PduR_DcmTransmit().
`PduInfoPtr`	Pointer to pointer to PduInfoStructure containing data pointer and length of a transmit buffer which is provided by the DCM.
`Length`	Minimum buffer size requested by the lower layer. A length of zero indicates that the length of the buffer can be of arbitrary size (larger than zero). The Length parameter is expected by DCM to be always zero.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: buffer has been successfully provided. BUFREQ_E_NOT_OK: no buffer provided, request failed. BUFREQ_E_BUSY: There is temporarily not enough data to be transmitted. Retry later.
Functional Description	
Called by a lower layer TP to request a buffer with data to be transmitted.	
Particularities and Limitations	
> ServiceID = 0x04 > Reentrant for different PduIds. Non reentrant for the same PduId. > This function is synchronous.	

Table 6-35 Dcm_ProvideTxBuffer ()

6.4.3.3.4 Dcm_TxConfirmation()

Prototype	
`void Dcm_TxConfirmation ( PduIdType DcmTxPduId, NotifResultType 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`	NTFRSLT_OK if the complete I-PDU has been transmitted. any other value: 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 transmission.	
Particularities and Limitations	
> ServiceID = 0x05 > This function is reentrant. > This function is synchronous.	

Table 6-36 Dcm_TxConfirmation()

6.4.4 CanTp

6.4.4.1 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 6-37 Dcm_ OnRequestDetection()

6.5 Configurable Interfaces

6.5.1 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:

6.5.1.1 <Module>_<DiagnosticService>()

Prototype	
`Std_ReturnType` **`<Module>_<DiagnosticService>`** `( Dcm_OpStatusType OpStatus, Dcm_MsgContextType* pMsgContext )`	
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
`pMsgContext`	Message-related information for one diagnostic protocol identifier. The pointers in pMsgContext points behind the SID.
Return code	
`Std_ReturnType`	E_OK: Request was successful. DCM_E_PENDING: Request 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_ProcessingDone for the current pMsgContext. Saves application code and stack usage.
Functional Description	
DCM calls a function of this kind as soon as a supported diagnostic 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/DcmDsdSidTabFnc.	
Particularities and Limitations	
> ServiceID = 0x32 > This function is reentrant. > This function is asynchronous.	

Table 6-38 <Module>_<DiagnosticService>()

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

Prototype	
`Std_ReturnType` **`<Module>_<DiagnosticService>_<SubService>`** `( Dcm_OpStatusType OpStatus, Dcm_MsgContextType* pMsgContext )`	
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.
`pMsgContext`	Message-related information for one diagnostic protocol sub-function identifier. The pointer in pMsgContext points behind the sub-function.
Return code	
`Std_ReturnType`	E_OK: Request was successful. DCM_E_PENDING: Request 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_ProcessingDone for the current pMsgContext. Saves application code and stack usage.
Functional Description	
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.	
Particularities and Limitations	
> ServiceID = 0x33 > This function is reentrant. > This function is asynchronous.	

Table 6-39 <Module>_<DiagnosticService>_<SubService>()

6.5.1.3 Dcm_SetProgConditions()

Prototype	
`Std_ReturnType` **`Dcm_SetProgConditions`** `( Dcm_ProgConditionsType * ProgConditions )`	
Parameter	
`ProgConditions`	Conditions on which the jump to bootloader 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 jumping to bootloader. The context parameters are defined in Dcm_ProgConditionsType.	
Particularities and Limitations	
> ServiceID = N/A > This function is not reentrant. > This function is asynchronous.	

Table 6-40 Dcm_SetProgConditions()

6.5.1.4 Dcm_GetProgConditions()

Prototype	
`Dcm_EcuStartModeType` **`Dcm_GetProgConditions`** `( Dcm_ProgConditionsType * ProgConditions )`	
Parameter	
`ProgConditions`	Conditions on which the jump from the bootloader has been requested.
Return code	
`Dcm_EcuStartModeType`	DCM_COLD_START: The ECU starts normally. DCM_WARM_START: The ECU starts from a bootloader jump. 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 ($50 or $51) has to be sent depending on request within the bootloader. The context parameters are defined in Dcm_ProgConditionsType.	
Particularities and Limitations	
> ServiceID = N/A > This function is not reentrant. > This function is synchronous.	

Table 6-41 Dcm_GetProgConditions()

6.5.1.5 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 responses 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	
> ServiceID = 41 > Not Reentrant > This function is synchronous.	

Table 6-42 Dcm_Confirmation()

6.5.1.6 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's 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 ($23)* - *ReadDataByIdentifier ($22)* (in case of Dynamical DID defined by memory address) - *ReadDataByPeriodicIdentifier ($2A)* (in case of Dynamical DID defined by memory address)	
Particularities and Limitations	
> ServiceID = 0x26 > This function is not reentrant. > This function is asynchronous.	

Table 6-43 Dcm_ReadMemory()

6.5.1.7 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's 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 service is needed for the implementation of UDS services : - *WriteMemoryByAddress ($3D)*	
Particularities and Limitations	
> ServiceID = 0x27 > This function is not reentrant. > This function is asynchronous.	

Table 6-44 Dcm_WriteMemory()

6.5.1.8 <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 *9.23 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 6-45 < Diagnostic Session Change Notification Callback >

6.5.1.9 <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 *9.16.2 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 6-46 <Security Access Change Notification Callback>

6.5.1.10 Dcm_GetRecoveryStates()

Prototype	
`Std_ReturnType Dcm_GetRecoveryStates (`*`Dcm_RecoveryInfoType*`*` RecoveryInfo)`	
Parameter	
`RecoveryInfo`	Contains all the information that has to 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 *9.27 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 in order 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 6-47 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.

6.5.1.11 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 particular 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 *9.29 How to Handle Multiple Diagnostic Service Variants*.	
Particularities and Limitations	
> This function is not reentrant. > This function is asynchronous.	

Table 6-48 Dcm_FilterDidLookUpResult

6.5.1.12 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 particular 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 *9.29 How to Handle Multiple Diagnostic Service Variants*.	
Particularities and Limitations	
> This function is not reentrant. > This function is asynchronous.	

Table 6-49 Dcm_FilterRidLookUpResult

6.5.2 Required Port Operation Functions

6.5.2.1 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 synchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 6-50 ConditionCheckRead()

6.5.2.2 ReadData() (asynchronous)

Prototype	
`Std_ReturnType` **`ReadData`** `( Dcm_OpStatusType OpStatus, uint8* 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 synchronous.	

Table 6-51 ReadData() (asynchronous)

6.5.2.3 ReadData() (synchronous)

Prototype	
`Std_ReturnType` **`ReadData`** `( uint8* 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_ASYNCH_FNC.	
Particularities and Limitations	
> ServiceID = 0x34 > This function is not reentrant. > This function is synchronous.	

Table 6-52 ReadData() (synchronous)

6.5.2.4 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 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 synchronous. > The "OpStatus" parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 6-53 ReadDataLength()

6.5.2.5 WriteData() (dynamic length)

Prototype	
`Std_ReturnType` **`WriteData`** `( uint8* 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 synchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 6-54 WriteData() (dynamic length)

6.5.2.6 WriteData() (static length)

Prototype	
`Std_ReturnType` **`WriteData`** `( uint8* 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 synchronous. > The "OpStatus" parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 6-55 WriteData() (static length)

6.5.2.7 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 *5.22.3 Implementation Aspects* of *InputOutputControlByIdentifier ($2F).*	
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 due to the fact that this operation shall always be executed synchronously. Refer to *5.22.3 Implementation Aspects* of *InputOutputControlByIdentifier ($2F)* for details. > The “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 6-56 ReturnControlToECU()

6.5.2.8 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 *5.22.3 Implementation Aspects* of *InputOutputControlByIdentifier ($2F).*	
Particularities and Limitations	
> ServiceID = 0x3C > 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 “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 6-57 ResetToDefault()

6.5.2.9 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 *5.22.3 Implementation Aspects* of *InputOutputControlByIdentifier ($2F).*	
Particularities and Limitations	
> ServiceID = 0x3A > 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 “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 6-58 FreezeCurrentState()

6.5.2.10 ShortTermAdjustment()

Prototype	
`Std_ReturnType` **`ShortTermAdjustment`** `( 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 *5.22.3 Implementation Aspects* of *InputOutputControlByIdentifier ($2F).*	
Particularities and Limitations	
> ServiceID = 0x3D > 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 “ControlMask” parameter is only available if DcmDspDidControlMask is set to “DCM_CONTROLMASK_EXTERNAL”.	

Table 6-59 ShortTermAdjustment()

6.5.2.11 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 synchronous. > The “OpStatus” parameter is only available if DcmDspDataUsePort is set to USE_DATA_ASYNCH_CLIENT_SERVER/ USE_DATA_ASYNCH_FNC.	

Table 6-60 GetScalingInformation()

6.5.2.12 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 synchronous.	

Table 6-61 Start()

6.5.2.13 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 > Not Reentrant > This function is synchronous.	

Table 6-62 Stop()

6.5.2.14 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 > Not Reentrant > This function is synchronous.

Table 6-63 RequestResults()

6.5.2.15 GetSeed() (with SADR)

Prototype	
`Std_ReturnType` **`GetSeed`** `(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 > Not Reentrant > This function is synchronous.	

Table 6-64 GetSeed() (with SADR)

6.5.2.16 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 > Not Reentrant > This function is synchronous.	

Table 6-65 GetSeed() (without SADR)

6.5.2.17 CompareKey()

Prototype	
`Std_ReturnType` **`CompareKey`** `( uint8* Key, Dcm_OpStatusType OpStatus [, Dcm_NegativeResponseCodeType* ErrorCode])`	
Parameter	
`Key`	Points to the requested key.
`OpStatus`	Status of the current operation.
`ErrorCode`	**Optional parameter.** Exists only in AR 4.2.1 or later enabled DCMs 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. Otherwise the DCM sends NRC 0x35/0x36 as for return value DCM_E_COMPARE_KEY_FAILED
Functional Description	
This function is a request from the DCM to the application to verify the requested security access level specific key.	
Particularities and Limitations	
> ServiceID = 0x47 > Not Reentrant > This function is synchronous.	

Table 6-66 CompareKey()

6.5.2.18 Indication()

Prototype	
`Std_ReturnType` **`Indication`** `( uint8 SID, uint8* RequestData, uint16 DataSize, uint8 ReqType, uint16 SourceAddress, Dcm_NegativeResponseCodeType* ErrorCode )`	
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.
`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_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 = N/A > Not Reentrant > This function is synchronous.	

Table 6-67 Indication()

6.5.2.19 Confirmation()

Prototype	
`Std_ReturnType` **`Confirmation`** `( uint8 SID, uint8 ReqType, uint16 SourceAddress, Dcm_ConfirmationStatusType ConfirmationStatus )`	
Parameter	
`SID`	Contains the diagnostic service Id.
`ReqType`	Specifies the diagnostic request type: 0 - physical request, 1 - functional request.
`SourceAddress`	Contains the diagnostic client source address.
`ConfirmationStatus`	Contains the response transmission resp. diagnostic response type.
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 = N/A > Not Reentrant > This function is synchronous.	

Table 6-68 Confirmation()

6.5.2.20 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 synchronous.	

Table 6-69 GetDTRValue()

6.5.2.21 RequestControl()

Prototype	
`Std_ReturnType` **`RequestControl`** `( uint8* OutBuffer, 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 synchronous.	

Table 6-70 RequestControl()

6.5.2.22 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 *5.8.4*). 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 *9.31 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 asynchronous.	

Table 6-71 GetInfotypeValueData()

6.5.2.23 StartProtocol()

Prototype	
`Std_ReturnType` **`StartProtocol`** `(`*`Dcm_ProtocolType`* `ProtocolID)`	
Parameter	
`ProtocolID`	Specifies the protocol ID of the new protocol to be started.
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 = N/A > This function is synchronous.	

Table 6-72 StartProtocol()

6.5.2.24 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 6-73 IsDidAvailable ()

6.5.2.25 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 synchronous.	

Table 6-74 ReadDidData()

6.5.2.26 WriteDidData()

Prototype	
`Std_ReturnType` **`WriteDidData`** `( uint16 DID, 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 synchronous.	

Table 6-75 WriteDidData()

6.5.2.27 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 > 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 6-76 GetSecurityAttemptCounter ()

6.5.2.28 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 changes, 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 > 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 6-77 SetSecurityAttemptCounter ()

6.5.2.29 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 currently available buffer size. 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 *9.24 How to Save RAM using Paged-Buffer for Large DIDs*.	
Particularities and Limitations	
> ServiceID = 0xA3 > This function is not reentrant. > This function is synchronous.	

Table 6-78 ReadData() (paged-data-reading)

6.6 Service Ports

6.6.1 Client-Server Interface

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

6.6.1.1 Provide Ports on DCM Side

At the Provide Ports of the DCM the API functions described in 6.2 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.

6.6.1.1.1 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()*	boolean Active, ERR{E_OK}
GetRequestKind	*Dcm_GetRequestKind()*	IN uint16 TesterSourceAddress, OUT Dcm_RequestKindType RequestKind, ERR{E_NOT_OK}

Table 6-79 DCMServices

6.6.1.2 Require Ports on DCM Side

At its Require Ports the DCM calls Operations. These Operations have to 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 *6.4.4*.

6.6.1.2.1 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 6-80 DataServices_<DataName>

6.6.1.2.2 RoutineServices_<RoutineName>

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

Table 6-81 RoutineServices_<RoutineName>

6.6.1.2.3 SecurityAccess_<SecurityLevelName>

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

Table 6-82 SecurityAccess_<SecurityLevelName>

6.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>

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

Table 6-83 ServiceRequestManufacturerNotification_<SWC>

6.6.1.2.5 ServiceRequestSupplierNotification_<SWC>

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

Table 6-84 ServiceRequestSupplierNotification_<SWC>

6.6.1.2.6 DtrServices_<MIDName>_<TIDName>

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

Table 6-85 DtrServices_<MIDName>_<TIDName>

6.6.1.2.7 RequestControlServices_<TIDName>

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

Table 6-86 RequestControlServices_<TIDName>

6.6.1.2.8 InfotypeServices_<VEHINFODATA>

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

Table 6-87 InfotypeServices_<VEHINFODATA>

6.6.1.2.9 CallbackDCMRequestServices_<SWC>

Operation	Callout
StartProtocol()	Rte_Call_CallbackDCMRequestServices_<SWC>_<Operation>

Table 6-88 CallbackDCMRequestServices_<SWC >

6.6.1.2.10 DataServices_DIDRange_<RangeName>

Operation	Callout
IsDidAvailable()	Rte_Call_DataServices_DIDRange_<RangeName>_<Operation>
ReadDidData()	
WriteDidData()	

Table 6-89 DataServices_DIDRange_<RangeName>

6.6.2 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.

Table 6-90 ModeDeclarationGroups managed by DCM

6.6.2.1 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 ($10)* or S3 timeout.

Table 6-91 DcmDiagnosticSessionControl callouts

Mode	Description
DefaultSession	Represents the UDS Default session (initial state).

Mode	Description
ProgrammingSession	Represents the UDS Programming session.
ExtendedSession	Represents the UDS Extended session.
</Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionRow> container's short-name	Any user defined session.

Table 6-92 DcmDiagnosticSessionControl modes

6.6.2.2 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 ($28)* or *DiagnosticSessionControl ($10)* or S3 timeout.

Table 6-93 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.

Mode	Description
DCM_DISABLE_RX_TX_NORM_NM	Reception and transmission of application and network management messages is disabled.

Table 6-94 DcmCommunicationControl_<ComM_CHANNEL_SNV> modes

6.6.2.3 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 ($11)* *EcuReset ($11)* or *DiagnosticSessionControl ($10)* processing. Invoked by *EcuReset ($11)* or *DiagnosticSessionControl ($10)* 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 ($11)* or *DiagnosticSessionControl ($10)* for bootloader related sessions

Table 6-95 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 6-96 DcmEcuReset modes

6.6.2.4 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 ($11)* processing. Invoked by *EcuReset ($11)*
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 ($11)*

Table 6-97 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 6-98 DcmModeRapidPowerShutDown modes

6.6.2.5 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 ($85), DiagnosticSessionControl ($10)* or S3 timeout.

Table 6-99 DcmControlDTCSetting callouts

Mode	Description
ENABLEDTCSETTING	DTC setting is enabled (initial state service 0x85 0x01 or *DiagnosticSessionControl ($10)* or S3 timeout)
DISABLEDTCSETTING	DTC setting is disabled (service 0x85 0x02)

Table 6-100 DcmControlDTCSetting modes

6.6.2.6 DcmSecurityAccess

FAQ
This mode declaration group is vendor specific one and only available under certain circumstances. Please refer to chapter *9.16 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 ($27)* or *DiagnosticSessionControl ($10)* or S3 timeout.

Table 6-101 DcmSecurityAccess callouts

Mode	Description
LockedLevel	Represents the UDS locked level (initial state).
</Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity/DcmDspSecurityRow > container's short-name	Any user defined security access level.

Table 6-102 DcmSecurityAccess modes

7 Configuration

7.1 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.

7.2 Configurable Attributes

The description of each configurable option is described within its online help in the Configurator 5 tool.

8 AUTOSAR Standard Compliance

8.1 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()*.

Table 8-1 Deviations to AUTOSAR

8.2 Additions/ Extensions

Additions/Extensions	Statement
DCM CPU peak load reduction support.	See *9.2*
RAM and run time optimization parameters for multi-client support	See *9.4*
Optimized DCM DEM iterator	DCM internal design.
Calibrateable configuration parameters	See *9.11*
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 4.1.2 API	See *9.13*
Combined OBD and UDS protocols over a single client connection	See *9.14*
Support for sub-functions 0x17, 0x18 and 0x19 of service Id 0x19.	See *ReadDiagnosticInformation ($19)*
Notification on security access level state change	See *9.16*
Integration within an AR 3.x environment	DCM will be delivered in a preconfigured state for interaction with BSWs implemented on the basis of the corresponding AR3 SWS.
Suppression on functional addressed requests	See *9.21*
Support of paged-buffer data access on DID signals	*See 9.24*
Configurable Security-Access level specific fixed bytes	*See 9.25*
Extensible keep-alive time period	*See 9.26*
DCM state recovery on reset /power on	*See 9.27*
Alternative solution for diagnostic service variant handling	*See 9.29*
Support for externally handled CEMR with more than four byte control mask.	According to see *[1]*, a CEMR can be only up to four bytes in size. MICROSAR DCM extends the SWS by allowing also any size of CEMR. Refer to *InputOutputControlByIdentifier ($2F)* for details.

Table 8-2 Additions/ Extensions to AUTOSAR

8.3 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 8-3 Limitations to AUTOSAR

9 Using the DCM

This chapter shall give some examples and hints, how to handle common use cases of the DCM.

9.1 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, 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 9-1 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 ($10)*) 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 meaningfull estimation).

Diagnostic Service	Buffer Size Calculation Type	
	Request	Response
RequestCurrentPowertrainDiagnosticData ($01)	Precise	Precise[1]
RequestPowertrainFreezeFrameData ($02)	Precise	Precise[2]
RequestEmissionRelatedDTC ($03)	Precise	Not estimated[3]
RequestOnBoardMonitorTestResults ($06)	Precise	Precise[4]
RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle ($07)	Precise	Not estimated[3]
RequestControlOfOnBoardSystemTestOrComponent ($08)	Precise	Precise
RequestVehicleInformation ($09)	Precise	Precise
RequestEmissionRelatedDTCsWithPermanentStatus ($0A)	Precise	Not estimated[3]
ReadDiagnosticInformation ($19)	Precise	Precise[5] Not estimated[3]
ReadDataByIdentifier ($22)	Precise[6]	Minimum estimation[7]
ReadMemoryByAddress ($23)	Precise[8]	Minimum estimation[9]
ReadScalingDataByIdentifier ($24)	Precise	Precise
SecurityAccess ($27)	Precise	Precise
ReadDataByPeriodicIdentifier ($2A)	Precise[6]	Precise[10]
DynamicallyDefineDataIdentifier ($2C)	Minimum estimation[11]	Precise
WriteDataByIdentifier ($2E)	Precise	Precise
InputOutputControlByIdentifier ($2F)	Precise	Precise
RoutineControl ($31)	Precise	Precise
WriteMemoryByAddress ($3D)	Minimum estimation[9]	Precise[8]
ControlDTCSetting ($85)	Precise	Precise

Table 9-1 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.
[4] Only if DCM knows the OBDMID configuration (refer to *9.30 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] UUDT buffer size is not considered here. Only the USDT request/response messages.
[11] It is guaranteed that at least one source item (DID or memory block) can be requested for the corresponding definition function. Subfunction „clear" is of course precisly calculated.

In some special cases like:

> Reading fault memory data:
  > *ReadDiagnosticInformation ($19)*
  > *RequestEmissionRelatedDTC ($03)*
  > *RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle ($07)*
  > *RequestEmissionRelatedDTCsWithPermanentStatus ($0A)*
> Reading multiple DIDs in a single request:
  > *ReadDataByIdentifier ($22)*

the required buffer size cannot be estimated or a pessimistic prediction will be applied in order to guarantee the ECU will always respond.

For the first case (*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.

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

The situation is different for the "*Reading multiple DIDs in a single request*" 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.

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 *9.24 How to Save RAM using Paged-Buffer for Large DIDs* for details on this usage.

9.2 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 in order to keep the P2 timings requirements. Additional 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 *ReadDiagnosticInformation ($19)*. Such services are:

> *ReadDataByIdentifier ($22)*
> *ReadDataByPeriodicIdentifier ($2A)*
> *DynamicallyDefineDataIdentifier ($2C)*

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 above mentioned diagnostic services.

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.

9.3 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]U[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.

9.4 How to Handle Multiple Diagnostic Clients Simultaneously

The 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 send always 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

9.5 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 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 ($10)*. If you need to restrict session activation by an ECU condition, you have to 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 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.

9.6 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 above mentioned use cases, 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 Configurator 5:

/Dcm/DcmConfigSet/DcmDsl/DcmDslServiceRequestManufacturerNotification

To set up a system supplier diagnostic service notification, add a configuration container in the 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

9.7 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

9.8 How to Jump into the FBL from Service DiagnosticSessionControl ($10)

The DCM provides means for transitions into the FBL from the ECU's application software. You can specify in the DCM configuration on which diagnostic session request this transition shall occur by the following parameter type:

/Dcm/DcmConfigSet/DcmDsp/DcmDspSession/DcmDspSessionRow/DcmDspSessionFor Boot

9.9 The HIS Compliant Jump into FBL

By default if a diagnostic request for SID $10 with a session Id configured for boot loader activation is received by the ECU, the DCM stores all necessary information for the FBL (via callout *Dcm_SetProgConditions()*)and resets the ECU without sending the final positive response to the request. This will be done by the FBL after the reset. Additionally, to avoid P2 time violation during the transition (reset) phase, the DCM can be configured to send a RCR-RP response prior resetting the ECU. For that purpose the DCM configuration parameter shall be set accordingly:

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmSendRespPendO nTransToBoot

9.9.1 The HIS Alternative Jump into FBL

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 achieve this behavior, you have to set the following DCM configuration parameter to TRUE:

/Dcm/DcmConfigSet/DcmGeneral/DcmResetToFblAfterSessionFinalResposeEnabled

9.10 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: $10 $01 and $11 $01.

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
TesterSourceAddr	Contains the Id of a tester that communications with the ECU over the communication bus shall be kept awaken while the non-default session is active. The tester Ids are assigned to each DCM connection in Configurator 5: /Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolRxTesterSourceAddr
ProtocolId	Not evaluated.
Sid	$10
SubFuncId	The Id of the session to be activated [$02 -$7E]. Must be a supported session within the DCM configuration (refer to *5.10DiagnosticSessionControl ($10)* for details).
ReprogrammingRequest	FALSE (Not evaluated.)
ApplUpdated	FALSE
ResponseRequired	FALSE

Table 9-2 Initialization of the Dcm_ProgConditionsType for non-default session activation at ECU power-on

9.11 How to Support Calibrateable 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.

9.11.1 OBD Calibration

FAQ
This feature is first supported in DCM 1.04.00.

In order to activate it, please use the following DCM ECUC parameter:
/Dcm/DcmConfigSet/DcmGeneral/DcmCalibrationOfObdIdsEnabled

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 *[6]*: PIDs/MIDs/TIDs/VIDs
  - For OBD in UDS resp. *[7]* and *[8]*:
    - DIDs in range 0xF400-0xF8FF
    - RIDs in range 0xE000-0xE1FF

9.11.1.1 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 in order 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 *9.11.1.2 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 calibrateable 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.

9.11.1.2 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 calibrateable DCM configuration objects described in *Table 9-3 Calibrateable OBD "availability parameter identifier" values*.

There are two types of OBD parameter identifiers:

> Availability Parameter Identifier (APID):
  > For *[6]*: 0x00, 0x20, 0x40, … 0xE0
  > For OBD in UDS resp. *[7]* and *[8]*: 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 always have to be consistent with the real ECU "data parameter identifier" availability configuration. To guarantee this consistency and simplify the calibration process, DCM uses calibrateable bitmaps for each "availability parameter identifier" that shall be supported.

The following table shows the overview of all OBD diagnostic service dependent calibrateable symbols:

Diagnostic Service ID	Table Name	Availability Condition
0x01	dcmCfg_Svc01SupportedIdMask[n] 1)	If SID 0x01 is to be supported.
0x02	dcmCfg_Svc02SupportedIdMask[8] 2)	If SID 0x02 is to be supported
0x06	dcmCfg_Svc06SupportedIdMask[n] 1)	If SID 0x06 is to be supported.
0x08	dcmCfg_Svc08SupportedIdMask[n] 1)	If SID 0x08 is to be supported.
0x09	dcmCfg_Svc09SupportedIdMask[n] 1)	If SID 0x09 is to be supported.
0x22	dcmCfg_Svc22SupportedIdMask[n] 3)	If SID 0x22 with any OBD DIDs is

Diagnostic Service ID	Table Name	Availability Condition
		to be supported.
0x31	dcmCfg_Svc31SupportedIdMask[n] [4)]	If SID 0x31 with any OBD RIDs is to be supported.

Table 9-3 Calibrateable 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 *[6]* 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 be always 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:

```
dcmCfg_Svc02SupportedIdMask [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 *9.11.1.1 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:

```
dcmCfg_Svc02SupportedIdMask [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:

```
dcmCfg_Svc22SupportedIdMask[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 have to be considered during the new bitmap values' generation:

- The APID concatenation has to be taken into account - see examples above how bit 0 of the corresponding APID masks changes.
- It is not possible to enable any APID or DPID that didn't 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 have to be filtered as described in *9.11.1.1 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**.

9.12 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*
- *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.

9.12.1 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) have to 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 9-6 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 9-4 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 a 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 9-4 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 9-5 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).	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).	Continue service processing (A), continue reception (B).	Continue response transmission (A), continue reception (B).	Do post-processing (A), continue reception (B).
Rx End	Send NRC 0x21 [3] (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	N/A	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	N/A	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)	N/A	Do post-processing (B), continue reception (A).	Do post-processing (B), start service processing (A).	N/A	N/A	N/A

Table 9-6 Protocol prioritization during non-default session

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 *9.4 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 on single frame request, or FC.OVFW in case of a multi-frame request.

9.12.2 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 in order 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.

9.12.3 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 of 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 ($04)*, 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).

9.13 How to Select DEM-DCM Interface Version

DCM is now (since version 2.01.00) capable of supporting both native DEM AR 4.0.3 and newer AR 4.1.2 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 DEM API version selection is introduced:/Dcm/DcmConfigSet/DcmGeneral/DcmDemApiVersion. For more details, please refer to the online help of this parameter.

9.14 How to Support OBD and UDS over a Single Client Connection

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 *9.12 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 short term 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 connection configured in DCM. If there are two or more connections, please use the multi-protocol prioritization mechanism with shared diagnostic buffers instead (refer to *9.12 How and When to Configure Multiple Protocols*).

9.15 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 the Vector Informatik company 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. In order to use already created user configuration file within the DCM's code generation process, you have to specify the full path to this file here:

/Dcm/DcmConfigSet/DcmGeneral/DcmUserConfigFile

9.16 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 the DCM is able to handle this internally:

> *ReadDataByPeriodicIdentifier ($2A)*

> *DynamicallyDefineDataIdentifier ($2C)*

For other diagnostic services, such as

> *InputOutputControlByIdentifier ($2F)* (will be automatically reset by DCM only on (re-)entering default session)

> *RoutineControl ($31)*

this task has to 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 ($10);*
  - TesterPresent Timeout;
  - Protocol Preemption;
- A successfully processed security unlocking sequence with service *SecurityAccess ($27)*

9.16.1 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.*

9.16.2 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 *6.5.1.9 <Security Access Change Notification Callback>.*

9.17 How to Deal with the PduR AR version

The DCM supports the interface to the PduR according to AR 3.x, AR 4.0.1, AR 4.0.3 and AR 4.1.2. Depending on the AR major version there are different configuration strategies.

9.17.1 AUTOSAR 3 Environment

If DCM shall interact with a PduR from an AR3 stack, then the delivery containing this DCM is already properly pre-configured. You cannot switch to any other PduR AR version.

9.17.2 AUTOSAR 4 Environment

For PduRs from AR 4.x stack, the concrete AR version is automatically derived from the PduR BSWMD file.

However, it is possible to enforce a specific AR version during integration by defining

MSR_PDUR_API_AR_VERSION in the file Compiler_Cfg.h.

Example: The PduR AR version is set to 4.0.3:

```
#define MSR_PDUR_API_AR_VERSION 0x403
```

For a detailed description of the PduR APIs, please refer to chapter *6.4.3 PduR.*

9.18 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 run-time - *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.

9.18.1 Post-build Variance Level

For all of the above mentioned supported variants, 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.

9.18.1.1 Communication Part

DCM's communication part includes every parameter located under the configuration container with path /Dcm/Dsl. 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 you cannot change 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.

9.18.1.2 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 "*9.29 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/Dsd and /Dcm/Dsp.

In general, the PBS support in DCM is limited only 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 in order 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 ($22)* 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 *9.18.1.2.1*.
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 has to implement only one data provider for the overlapping UDS and OBD entities.

Table 9-7 Post-build configuration rules on invariant DCM parameters

9.18.1.2.1 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 have to be taken into account.

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 *10.2 Code Generation Time Messages*.

Only one kind of diagnostic entity will not be validated upon execution precondition mismatch: **memory ranges**.

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.

> A diagnostic entity has execution precondition that refers to states not existing in some variants.

> > A special case: The state group (e.g. SecurityAccess) is not available in all variants, since service *SecurityAccess ($27)* is not available at all in those variants.

In the described case the affected diagnostic entity will be configured with an empty list of precondition related states. A list with no state references is always interpreted as “no execution precondition restrictions”. This of course mismatches the original semantic of the precondition: “diagnostic entity accessible ***_only_*** in the referenced state(s)”.

Solution:

Having a diagnostic entity available in a variant where it shall not be executed in any (remaining) state of a state group sounds implausible. Actually such a diagnostic entity (e.g. diagnostic service, DID etc.) will never be able to send a positive response and thus shall not even exist in the affected variant(s).

Example:

Diagnostic service shall be supported only in the programming session. This service is configured to be available in a variant, where the programming session is not available at all. As a result the given service shall not exist in the variant too.

What happens if the affected diagnostic entity is not removed from the variant?

DCM will interpret the precondition as “there is no precondition” and will merge these states over all the variants, allowing the diagnostic entity to be always accessible.

> The execution precondition depends on the preconditions of other related diagnostic entities.

In order 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 ($22)* 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 ($22)* to be accessible in the default session does not exist in a new variant where other readable DIDs are still available, meaning service *ReadDataByIdentifier ($22)* 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 no 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.

9.18.2 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()* has to 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 EcuM based on the post-build configuration. If no MICROSAR EcuM is used, the procedure of how to find the proper initialization pointers is out of scope of this document.

9.18.2.1 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 9-8.*

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.
`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 9-8 Error Codes possible during Post-Build initialization failure

If no MICROSAR EcuM is used, this error hooks and the error code constants have to 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.

9.18.3 Post-build Variants

9.18.3.1 Post-build selectable

The MICROSAR Identity Manager (refer to *[10]*) 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 *9.18.2 Initialization*).

9.18.3.2 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 of 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 *9.18.3.3*).
  - Greater risks of passing an invalid pointer during module initialization time.

For details about the post-build loadable feature, please refer to *[9]*.

9.18.3.3 Post-build loadable selectable

This variant actually 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 *9.18.3.1* and *9.18.3.2*.

9.18.3.4 Post-build deleteable

This variant is actually 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 *[9]*.

9.19 Handling with DID Ranges

9.19.1 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 is able to support 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.

9.19.2 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 ($2C)* service will validate the source position and size parameters only upon the configured DID range maximum possible length (*9.19.3 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 ($22)*, 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 in order 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 (*9.24 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).

9.19.3 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 ($22)* and *WriteDataByIdentifier ($2E)* 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.

9.20 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 has to 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 has to 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.

9.21 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 has to be configured to "DCM_NET_ADDR_MIXED_11":

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslAddressingType

9.22 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

9.23 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 the DCM is able to handle this internally:

> *ReadDataByPeriodicIdentifier ($2A)*
> *DynamicallyDefineDataIdentifier ($2C)*
> *CommunicationControl ($28)*
> *ControlDTCSetting ($85)*

For other diagnostic services, such as

> *InputOutputControlByIdentifier ($2F)* (will be automatically reset by DCM only on (re-)entering default session)
> *RoutineControl ($31)*

this task has to 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 *6.5.1.8 <Diagnostic Session Change Notification Callback>*.

9.24 How to Save RAM using Paged-Buffer for Large DIDs

9.24.1 Introduction

According to all up to now released AUTOSAR DCM SWS documents, the only service that supports paged-data reading is service *ReadDiagnosticInformation ($19)*.

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 has to 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 DCM offers a concept for paged-data reading, described in details further below.

9.24.2 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 has to 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 as long as they are applicable for that concept.

There are also some possible drawbacks that have to 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.

9.24.3 Implementation Limitations

When paged-data access of a DID is intended to be used, there are still some limitations that have to 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 makes also no real sense to apply this concept.
- Paged-data access cannot be used for DIDRanges (see *9.19 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.

9.24.4 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*

9.24.4.1 Straightforward DID Paged-Data Reading

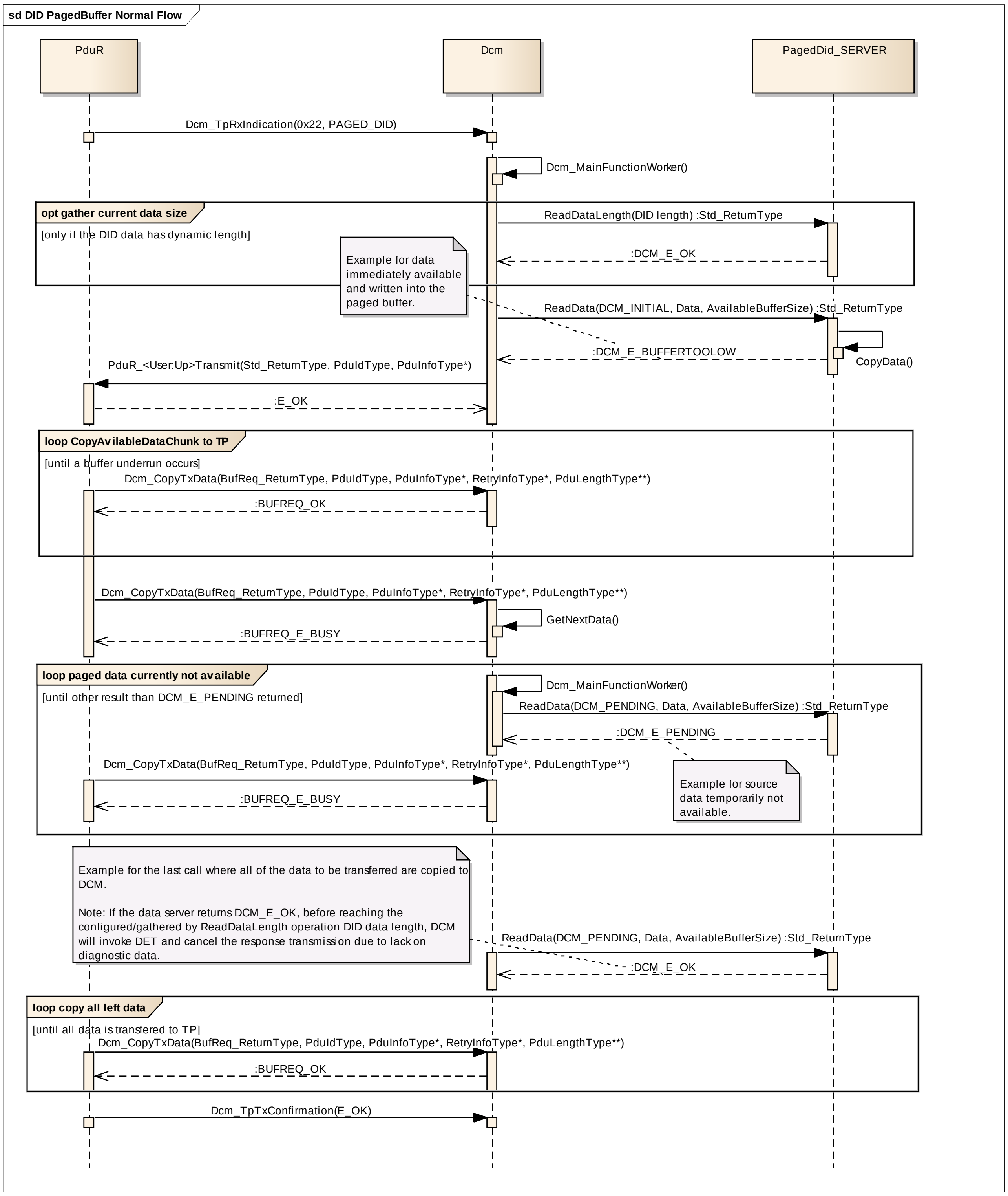
sd DID PagedBuffer Normal Flow
PduR
Dcm
PagedDid_SERVER
Dcm_TpRxIndication(0x22, PAGED_DID)
Dcm_MainFunctionWorker()
opt gather current data size
[only if the DID data has dynamic length]
ReadDataLength(DID length) :Std_ReturnType
:DCM_E_OK
Example for data immediately available and written into the paged buffer.
ReadData(DCM_INITIAL, Data, AvailableBufferSize) :Std_ReturnType
CopyData()
:DCM_E_BUFFERTOOLOW
PduR_<User:Up>Transmit(Std_ReturnType, PduIdType, PduInfoType*)
:E_OK
loop CopyAvilableDataChunk to TP
[until a buffer underrun occurs]
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
:BUFREQ_OK
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
GetNextData()
:BUFREQ_E_BUSY
loop paged data currently not available
[until other result than DCM_E_PENDING returned]
Dcm_MainFunctionWorker()
ReadData(DCM_PENDING, Data, AvailableBufferSize) :Std_ReturnType
:DCM_E_PENDING
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
:BUFREQ_E_BUSY
Example for source data temporarily not available.
Example for the last call where all of the data to be transferred are copied to DCM.
Note: If the data server returns DCM_E_OK, before reaching the configured/gathered by ReadDataLength operation DID data length, DCM will invoke DET and cancel the response transmission due to lack on diagnostic data.
ReadData(DCM_PENDING, Data, AvailableBufferSize) :Std_ReturnType
:DCM_E_OK
loop copy all left data
[until all data is transfered to TP]
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
:BUFREQ_OK
Dcm_TpTxConfirmation(E_OK)

Figure 9-1 Straightforward DID paged-data reading

9.24.4.2 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.

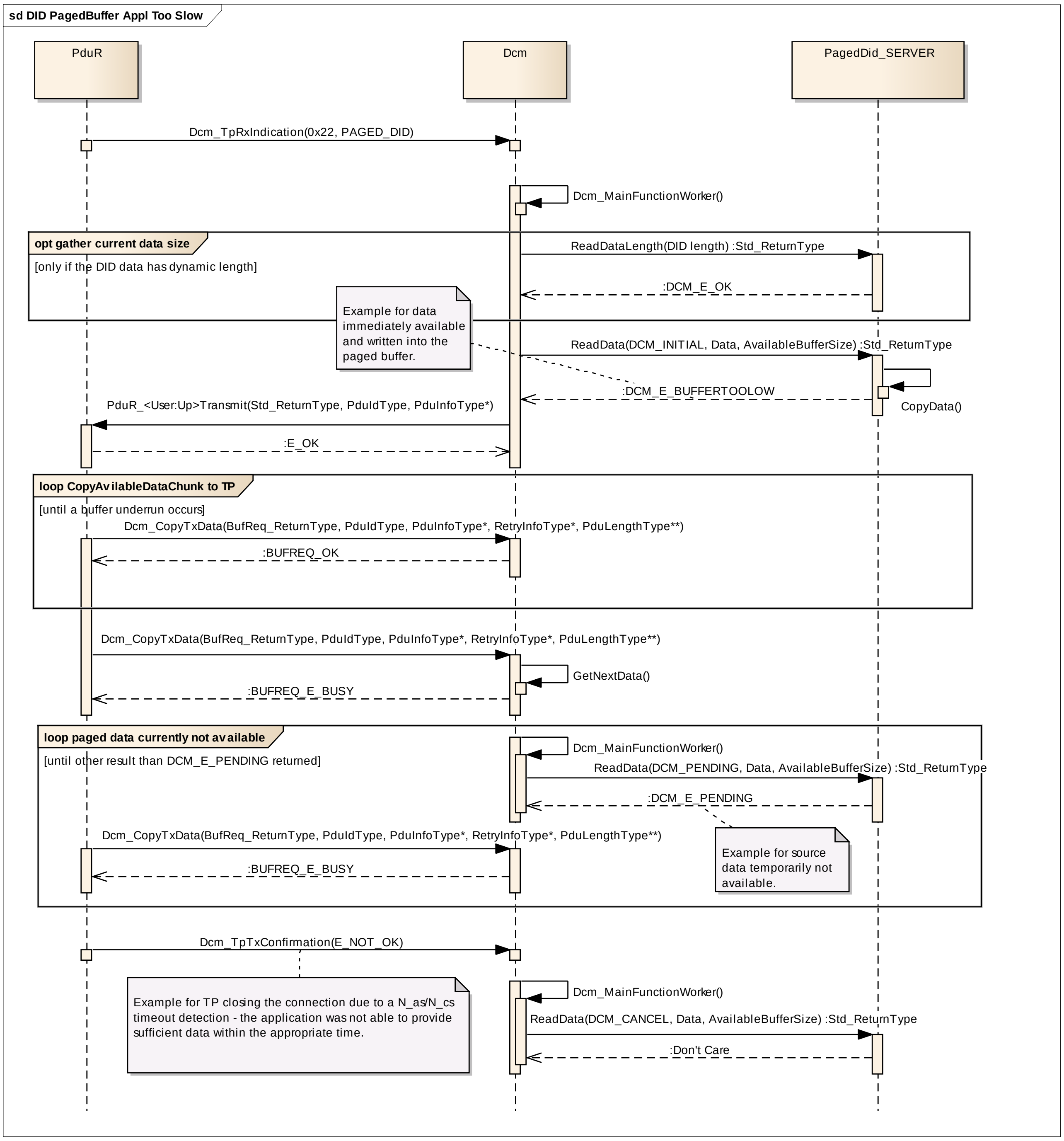
sd DID PagedBuffer Appl Too Slow
PduR
Dcm
PagedDid_SERVER
Dcm_TpRxIndication(0x22, PAGED_DID)
Dcm_MainFunctionWorker()
opt gather current data size
[only if the DID data has dynamic length]
ReadDataLength(DID length) :Std_ReturnType
:DCM_E_OK
Example for data immediately available and written into the paged buffer.
ReadData(DCM_INITIAL, Data, AvailableBufferSize) :Std_ReturnType
CopyData()
:DCM_E_BUFFERTOOLOW
PduR_<User:Up>Transmit(Std_ReturnType, PduIdType, PduInfoType*)
:E_OK
loop CopyAvilableDataChunk to TP
[until a buffer underrun occurs]
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
:BUFREQ_OK
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
GetNextData()
:BUFREQ_E_BUSY
loop paged data currently not available
[until other result than DCM_E_PENDING returned]
Dcm_MainFunctionWorker()
ReadData(DCM_PENDING, Data, AvailableBufferSize) :Std_ReturnType
:DCM_E_PENDING
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
:BUFREQ_E_BUSY
Example for source data temporarily not available.
Dcm_TpTxConfirmation(E_NOT_OK)
Example for TP closing the connection due to a N_as/N_cs timeout detection - the application was not able to provide sufficient data within the appropriate time.
Dcm_MainFunctionWorker()
ReadData(DCM_CANCEL, Data, AvailableBufferSize) :Std_ReturnType
:Don't Care

Figure 9-2 DID paged-data reading cancelled due to TP layer transmission abortion

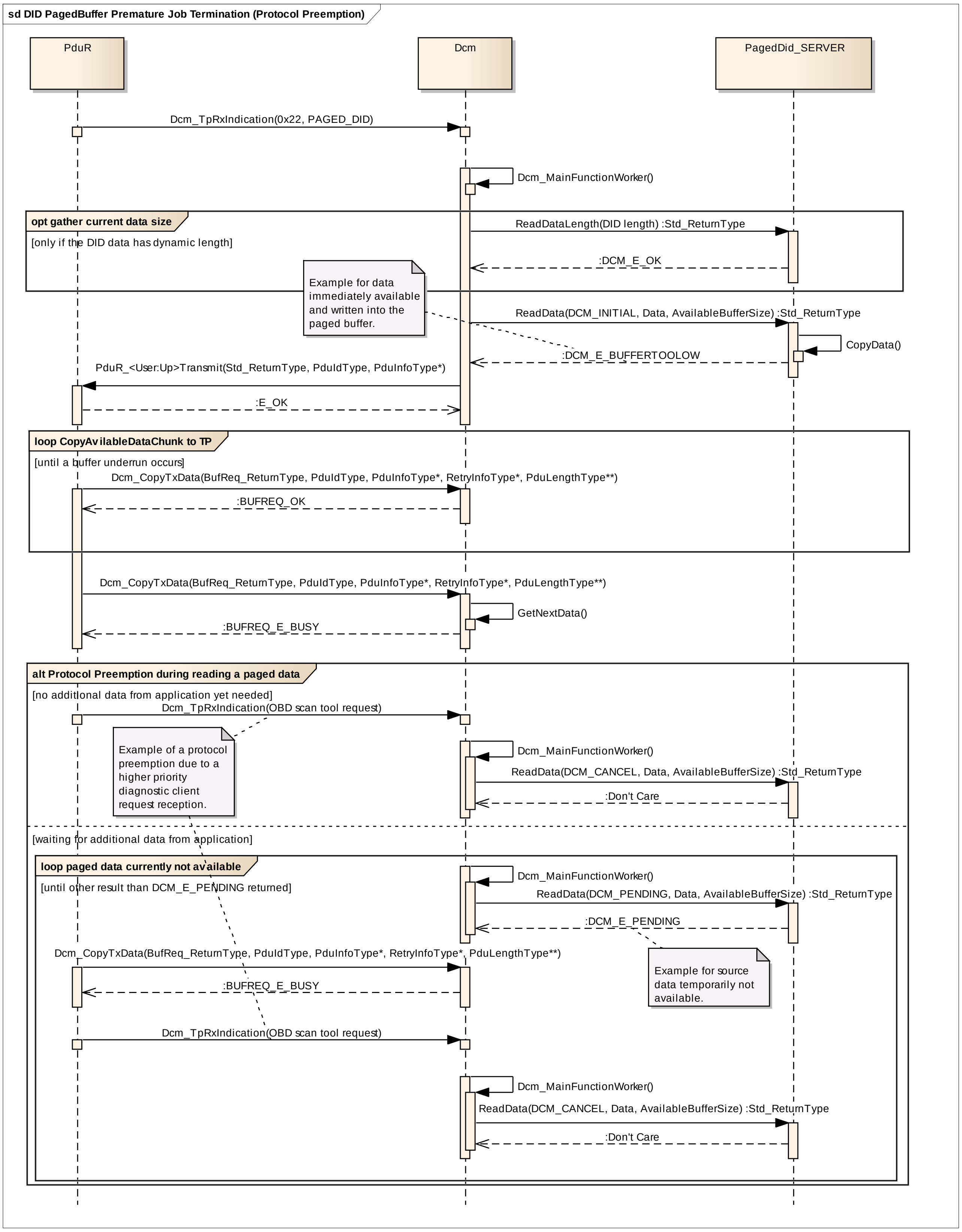
sd DID PagedBuffer Premature Job Termination (Protocol Preemption)
PduR
Dcm
PagedDid_SERVER
Dcm_TpRxIndication(0x22, PAGED_DID)
Dcm_MainFunctionWorker()
opt gather current data size
[only if the DID data has dynamic length]
ReadDataLength(DID length) :Std_ReturnType
:DCM_E_OK
Example for data immediately available and written into the paged buffer.
ReadData(DCM_INITIAL, Data, AvailableBufferSize) :Std_ReturnType
CopyData()
:DCM_E_BUFFERTOOLOW
PduR_<User:Up>Transmit(Std_ReturnType, PduIdType, PduInfoType*)
:E_OK
loop CopyAvilableDataChunk to TP
[until a buffer underrun occurs]
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
:BUFREQ_OK
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
GetNextData()
:BUFREQ_E_BUSY
alt Protocol Preemption during reading a paged data
[no additional data from application yet needed]
Dcm_TpRxIndication(OBD scan tool request)
Example of a protocol preemption due to a higher priority diagnostic client request reception.
Dcm_MainFunctionWorker()
ReadData(DCM_CANCEL, Data, AvailableBufferSize) :Std_ReturnType
:Don't Care
[waiting for additional data from application]
loop paged data currently not available
[until other result than DCM_E_PENDING returned]
Dcm_MainFunctionWorker()
ReadData(DCM_PENDING, Data, AvailableBufferSize) :Std_ReturnType
:DCM_E_PENDING
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, PduInfoType*, RetryInfoType*, PduLengthType**)
Example for source data temporarily not available.
:BUFREQ_E_BUSY
Dcm_TpRxIndication(OBD scan tool request)
Dcm_MainFunctionWorker()
ReadData(DCM_CANCEL, Data, AvailableBufferSize) :Std_ReturnType
:Don't Care

Figure 9-3 Protocol preemption during DID paged-data access

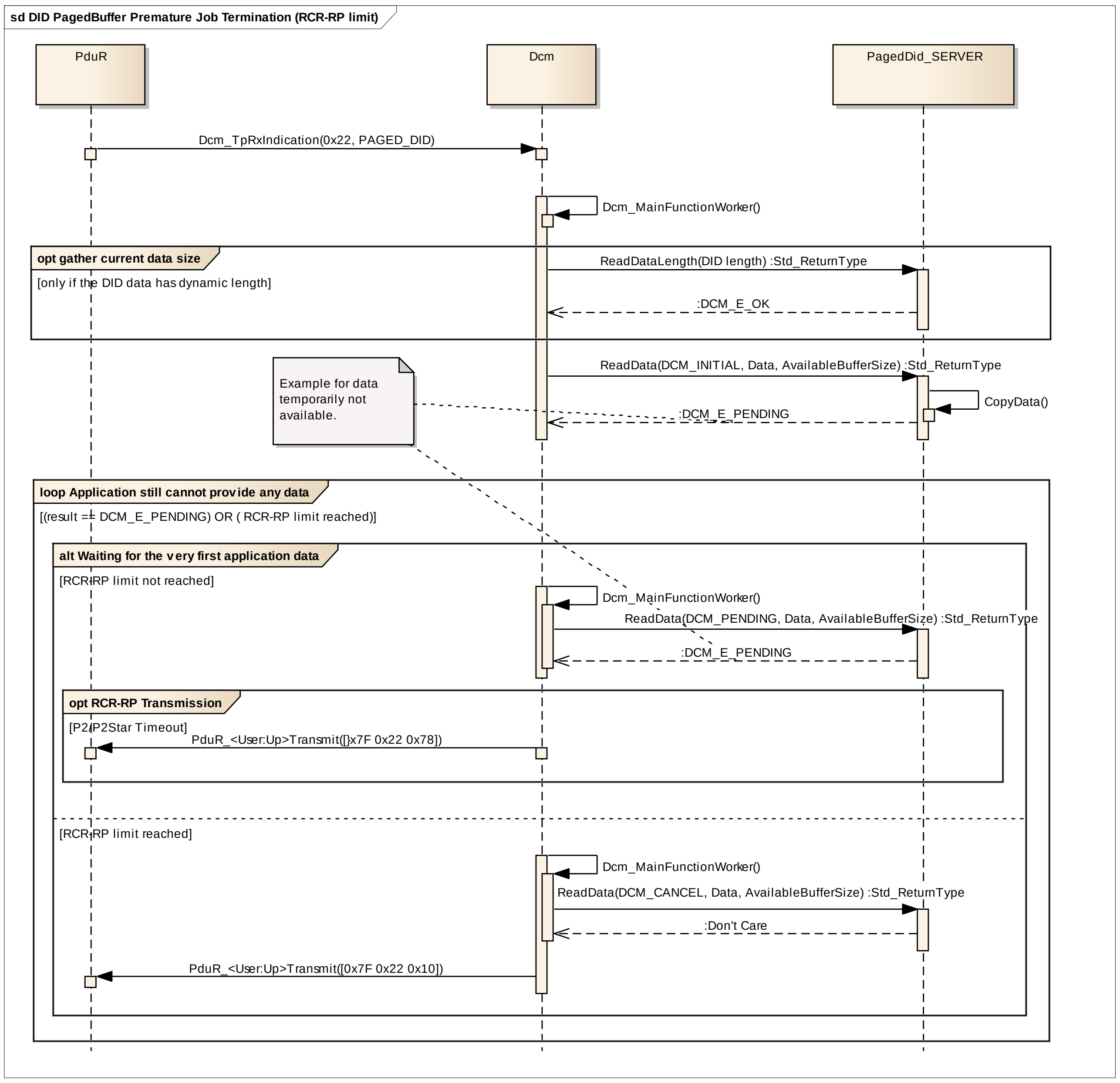
sd DID PagedBuffer Premature Job Termination (RCR-RP limit)
PduR
Dcm
PagedDid_SERVER
Dcm_TpRxIndication(0x22, PAGED_DID)
Dcm_MainFunctionWorker()
opt gather current data size
[only if the DID data has dynamic length]
ReadDataLength(DID length) :Std_ReturnType
:DCM_E_OK
Example for data temporarily not available.
ReadData(DCM_INITIAL, Data, AvailableBufferSize) :Std_ReturnType
CopyData()
:DCM_E_PENDING
loop Application still cannot provide any data
[(result == DCM_E_PENDING) OR (RCR-RP limit reached)]
alt Waiting for the very first application data
[RCR-RP limit not reached]
Dcm_MainFunctionWorker()
ReadData(DCM_PENDING, Data, AvailableBufferSize) :Std_ReturnType
:DCM_E_PENDING
opt RCR-RP Transmission
[P2/P2Star Timeout]
PduR_<User:Up>Transmit([]x7F 0x22 0x78])
[RCR-RP limit reached]
Dcm_MainFunctionWorker()
ReadData(DCM_CANCEL, Data, AvailableBufferSize) :Std_ReturnType
:Don't Care
PduR_<User:Up>Transmit([0x7F 0x22 0x10])

Figure 9-4 RCR-RP limit reached during DID paged-data access

9.24.5 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 has to 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

9.25 How to Get Security-Access Level Specific Fixed Byte Values

9.25.1 Introduction

In some ECU projects it is desired, that the 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 DCM configuration. In that way, both diagnostic client and server (ECU) have always the correct values.

To achieve this goal, MICROSAR 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()*.

9.25.2 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 newly introduced provided port operation *Dcm_GetSecurityLevelFixedBytes()*.

9.25.3 Configuration Aspects

If a security level shall provide a fixed bytes 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.

9.26 How to Extend the Diag Keep Alive Time during Diagnostics

9.26.1 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 period 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 period 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.

9.26.2 Configuration Aspects

If such an extended time period 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

9.27 How to Recover DCM State Context on ECU Reset/ Power On

9.27.1 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).

9.27.2 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 have to 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 in order to avoid any unnecessary fault memory entries.

9.27.3 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

9.28 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. In order to configure such a connection, you have to delete the following sub-container of it:

/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslProtocolTx

9.29 How to Handle Multiple Diagnostic Service Variants

9.29.1 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 is capable of using DCM tools to update the configuration at run-time 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.

9.29.2 Filtering Level Availability and the Corresponding Filtering Tools

In the following two tables, namely, *Table 9-9* and *Table 9-10*, 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 9-9 Filtering level availability

In order to get an advantage of DCM extended filtering tools, the extended filtering feature has to be activated in the configuration tools. Refer to *Table 9-10* under column "Configuration Aspects" for more details.

Filtered Diagnostic Object	Filtering API / Callback	Configuration Aspects
Service, Sub-service (Sub-function)	Refer to: *6.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>* <Operation> = *Indication()*	Refer to: *9.6 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: *6.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>* <Operation> = *Indication()*	Refer to: *9.6 How to Get Notified on a Diagnostic Service Execution Start and End*
Memory, Memory Operation	Refer to: *6.6.1.2.4 ServiceRequestManufacturerNotification_<SWC>* <Operation> = *Indication()*	Refer to: *9.6 How to Get Notified on a Diagnostic Service Execution Start and End*
PID	Refer to: *9.29.3 Filtering OBD Objects*	Refer to: *9.29.3 Filtering OBD Objects*
MID		
TID		
VID		

Table 9-10 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 run-time 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.

9.29.3 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 *9.11.1.2*). 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.

9.29.3.1 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 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 the purpose of multiple-variant-handling, so they don’t 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 9-3 Calibrateable 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.

9.30 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 ($06)* 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.

9.30.1 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 will have to 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 of 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.

9.30.2 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 10-1   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

9.31 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.

9.31.1 Implementation Limitations

While the VID reading via *RequestVehicleInformation ($09)* is not really affected by the API change, *ReadDataByIdentifier ($22)* 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 *9.19 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 *5.8.4 Configuration Aspects*.

9.32 How to setup DCM for Sender-Receiver Communication

Additionally to the *Client-Server Interface* type of communication with the application, starting with DCM 7.00.00 also the Sender-Receiver kind is supported for the following diagnostic services only:

- Data Identifier (DID) related:
  - Read Access:
    - *ReadDataByIdentifier ($22)*
    - *ReadDataByPeriodicIdentifier ($2A)*
  - Write Access:
    - *WriteDataByIdentifier ($2E)*
  - IO Control:
    - *InputOutputControlByIdentifier ($2F)* (first available in DCM 7.01.00)

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.

9.32.1 Implementation Limitations

When using the DCM S/R communication some limitations and particularities shall be considered:

- 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 read and write operations, the supported elements' base data types are:
  - Atomics: (u|s)int(8|16|32)
  - Fields: (u|s)int8
- For the DID related IOControl operations, the supported element data types are:
  - Atomics: uint(8|16|32)
  - Fields: uint8
  - CEMR: 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.

9.32.2 Application usage Scenario

In order to get S/R IOControl operation working with your application, the following design aspects shall be considered:

> On diagnostic request for service *InputOutputControlByIdentifier ($2F)*

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, read carefully the specifics of the CEMR handling in the corresponding chapter *InputOutputControlByIdentifier ($2F)*).

> 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!

9.32.3 Configuration Aspects

> In order to enable S/R communication on DIDs, you have to specify the RTE usage on the corresponding DID data elements to be SENDER_RECEIVER:

/Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort

> Additionally, if the S/R communication shall be applied on DID level (i.e. all DID data elements will be merged into a single data block with the total length of the DID), then following parameter shall be set accordingly:

/Dcm/DcmConfigSet/DcmDsp/DcmDspDid/DcmDspDidUsePort

For usage details of these particular parameters, please refer to the Configurator5 online help.

9.33 How to Support Routine Info Byte with UDS RIDs

9.33.1 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 ($31)* is requested. The DCM provides a means to report this Routine Info Byte without need of application intervention.

9.33.2 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 has to provide the Routine Info Byte if needed.

10 Troubleshooting

10.1 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 ($2A)* 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*).
Unsupported PduR version!	Unrecognized/unsupported PduR version is specified.	Refer to *9.17 How to Deal with the PduR AR version*.
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 DEM. - In a MICROSAR DEM no extended data records are defined. - In a MICROSAR DEM no DTC refers an extended data record.	- Refer to *5.13.3.1Reporting Stored DTC Environment Data* for information about this configuration. - Correct the MICROSAR DEM configuration. - Remove the corresponding DCM *ReadDiagnosticInformation ($19)* 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 DEM.	- Refer to *5.13.3.1Reporting Stored DTC Environment Data* for information about this configuration. - Correct the MICROSAR DEM

Error Message	Reason	Countermeasure
	- In a MICROSAR 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.	configuration. - Remove the corresponding DCM *ReadDiagnosticInformation ($19)* 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 *9.13 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 ($2A)*)
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 this job too or vice-versa.	Refer to the *9.30 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 *9.24.3 Implementation Limitations* for details on using paged DIDs.
Paged DIDs are not allowed if any OBD2 VIDs as per AR4.2.2 are enabled!	Incompatible features have been activated.	Refer to *9.31.1 Implementation Limitations* for details on using OBD2 VIDs with AR 4.2.2 API.
Any other message	Internal inconsistency detection.	Contact Vector.

Table 10-1 Compile time error messages

10.2 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 *9.18.1.2.1 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	An **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.	
DCM05025	A **diagnostic sub-service** has been	

Message ID	Reason	Description
	defined to have different execution preconditions for the state group "**security access**" in multiple variants.	

Table 10-2 Code Generation Time Messages

11 Glossary and Abbreviations

11.1 Glossary

Term	Description
Configurator 5	Configuration and generation tool for MICROSAR components

Table 11-1 Glossary

11.2 Abbreviations

Abbreviation	Description
ALFID	Address and Length Format Identifier
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
C/S	Client/Server (Port)
CDD	Complex Device Driver
CEM	Control Enable Mask
CEMR	CEM Record
DCM	Diagnostic Communication Manager
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
FC.OVFW	Flow Control with status Overflow
FBL	Flash Boot Loader
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MID	Monitor Identifier
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
SRS	Software Requirement Specification
STRT	Service To Respond To
SWC	Software Component
SWS	Software Specification
TID	Test Identifier
VID	Vehicle Identification Number

Table 11-2 Abbreviations

12 Contact

Visit our website for more information on

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

www.vector.com