

Document Title	Specification of Diagnostic Communication Manager
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Identification No	018
Document Classification	Standard

Document Version	3.0.0
Document Status	Final
Part of Release	3.0
Revision	0001

Document Change History			
Date	Version	Changed by	Change Description
28.11.2007	3.0.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Rework of the interfaces with RTE (remove of Central Diagnostic SWC concept) • Correction of issues identified on R2.1 • Document meta information extended • Small layout adaptations made
24.01.2007	2.1.1	AUTOSAR Administration	<ul style="list-style-type: none"> • “Advice for users” revised • “Revision Information” added
05.12.2006	2.1.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Corrections in configuration chapter • Rework on interface between DCM and DEM according to changes in DEM SWS • Corrections in Sequence diagram • Addition of header files inclusions • Legal disclaimer revised
29.06.2006	2.0.1	AUTOSAR Administration	Layout Adaptations
26.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template. <ul style="list-style-type: none"> • Major changes in chapter 10 • Structure of document changed partly • Other changes see chapter 11
10.07.2005	1.0.0	AUTOSAR Administration	Initial release

Page left intentionally blank

Disclaimer

Any use of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2007 AUTOSAR Development Partnership. All rights reserved.

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1	Introduction and functional overview	10
1.1	DCM in AUTOSAR architecture	12
2	Acronyms and abbreviations	13
3	Related documentation.....	15
3.1	Input documents.....	15
3.2	Related standards and norms	15
4	Constraints and assumptions	17
4.1	Limitations	17
4.2	Applicability to car domains.....	19
4.3	Applicability to emission related environments (OBD).....	19
5	Dependencies to other modules.....	20
5.1	File structure	22
5.1.1	Code file structure.....	22
5.1.2	Header file structure	23
6	Requirements traceability	24
7	Functional specification	27
7.1	Overview	27
7.1.1	DCM Architecture overview	28
7.2	Functional blocks.....	29
7.2.1	DSL (Diagnostic Session Layer).....	29
7.2.1.1	Introduction.....	29
7.2.1.2	Use cases.....	29
7.2.1.3	Interaction with other modules.....	29
7.2.1.4	Functional Description	30
7.2.1.4.1	Overview:	30
7.2.1.4.2	Detailed description	30
7.2.1.4.2.1	Forward requests from PDU Router to DSD	30
7.2.1.4.2.2	Concurrent “TesterPresent” (“keep alive logic”)	31
7.2.1.4.2.3	Forward responses from DSD to PDU Router.....	33
7.2.1.4.2.4	Guarantee timing to tester by sending busy responses.....	33
7.2.1.4.2.5	Support of periodic transmission	34
7.2.1.4.2.6	Support of ROE transmission.....	34
7.2.1.4.2.7	Support of segmented response (“PagedBuffer”).....	35
7.2.1.4.2.8	Support of ResponsePending response triggered by application	36
7.2.1.4.2.9	Manage security level.....	36
7.2.1.4.2.10	Manage session state	37
7.2.1.4.2.11	Keep track of active non-default sessions.....	37
7.2.1.4.2.12	Informs depending modules on session change	37
7.2.1.4.2.13	Allow to modify timings.....	38
7.2.1.4.2.14	Handling of different diagnostic protocols.....	39

7.2.1.4.2.15	Manage resources.....	40
7.2.1.4.2.16	Communication Mode Handling	41
7.2.2	DSD (Diagnostic Service Dispatcher)	43
7.2.2.1	Introduction.....	43
7.2.2.2	Use cases.....	43
7.2.2.3	Interaction with other modules.....	45
7.2.2.4	Functional Description	46
7.2.2.4.1	Support check of diagnostic service identifier and adaptation of the Diagnostic message	46
7.2.2.4.2	Handling of „suppressPosRspMsgIndicationBit“	48
7.2.2.4.3	Verification functionality	48
7.2.2.4.4	Distribution of Diagnostic Message to data processor (DSP or AUTOSAR Software Component)	50
7.2.2.4.4.1	Assemble positive or negative response	50
7.2.2.4.5	Initiate transmission	52
7.2.3	DSP (Diagnostic Service Processing).....	52
7.2.3.1	Introduction.....	52
7.2.3.2	Use cases.....	54
7.2.3.3	Interaction with other modules.....	54
7.2.3.4	Sub-function and format check.....	54
7.2.3.5	Error manager interface.....	55
7.2.3.5.1	Usage of Paged Buffer mechanism	56
7.2.3.5.2	Basic sequence.....	56
7.2.3.5.3	Emission related diagnostic services according ISO15031-5.....	57
7.2.3.5.4	Enhanced diagnostic services according ISO14229-1 (UDS).....	61
7.2.3.6	Diagnostic communication and security	70
7.2.3.6.1	Diagnostic Session Control.....	70
7.2.3.6.2	Tester Present	71
7.2.3.6.3	SecurityAccess	71
7.3	Error classification.....	73
7.4	Error detection.....	74
7.5	Error notification	74
8	API specification.....	75
8.1	General	75
8.1.1	Interface description of the DCM Module.....	75
8.2	Imported types.....	77
8.2.1	Standard types.....	77
8.2.2	Other types	77
8.3	Type definitions	78
8.3.1	DCM Types.....	78
8.3.1.1	Dcm_StatusType	78
8.3.1.2	Dcm_ProtocolType	79
8.3.1.3	Dcm_MsgItemType	79
8.3.1.4	Dcm_MsgType	80
8.3.1.5	Dcm_MsgLenType	80
8.3.1.6	Dcm_MsgAddInfoType	80
8.3.1.7	Dcm_IdContextType.....	80
8.3.1.8	Dcm_MsgContextType	80
8.3.1.9	Dcm_NegativeResponseCodeType	82

8.3.2	DSL (Diagnostic Session Layer) Types	87
8.3.2.1	Dcm_TimerServerType	87
8.3.2.2	Dcm_TimerModeType	88
8.3.2.3	Dcm_SesCtrlType	88
8.3.2.4	Dcm_SecLevelType	88
8.3.2.5	Dcm_ConfirmationStatusType	88
8.4	Function definitions	89
8.4.1	DCM Functions	89
8.4.1.1	Dcm_Init	89
8.4.1.2	Dcm_GetVersionInfo	89
8.4.2	DSL (Diagnostic Session Layer) Functions	90
8.4.2.1	Dcm_GetSecurityLevel	90
8.4.2.2	Dcm_GetSesCtrlType	91
8.4.2.3	Dcm_GetActiveProtocol	91
8.4.3	DSD (Diagnostic Service Dispatcher)	91
8.5	Call-back notifications	91
8.5.1	DSL (Diagnostic Session Layer) Functions	92
8.5.1.1	Dcm_ProvideRxBuffer	92
8.5.1.2	Dcm_RxIndication	93
8.5.1.3	Dcm_ProvideTxBuffer	93
8.5.1.4	Dcm_TxConfirmation	95
8.5.1.5	Dcm_ComM_NoComModeEntered	96
8.5.1.6	Dcm_ComM_SilentComModeEntered	96
8.5.1.7	Dcm_ComM_FullComModeEntered	96
8.6	Scheduled functions	97
8.6.1	Dcm_MainFunction	97
8.7	Expected Interfaces	97
8.7.1	Mandatory Interfaces	97
8.7.2	Optional Interfaces	99
8.7.3	RTE interfaces	99
8.7.3.1	DSL (Diagnostic Session Layer) Functions	99
8.7.3.1.1	Dcm_GetSesChgPermission	99
8.7.3.1.2	Dcm_SesCtrlChangeIndication	100
8.7.3.1.3	Dcm_ConfirmationRespPend	100
8.7.3.1.4	Dcm_StartProtocol	101
8.7.3.1.5	Dcm_StopProtocol	101
8.7.3.1.6	Dcm_Indication	101
8.7.3.2	Interface for special services	102
8.7.3.2.1	Dcm_GetSeed	102
8.7.3.2.2	Dcm_CompareKey	103
8.7.4	Configurable interfaces	103
8.7.4.1	Interfaces with RTE	103
8.7.4.1.1	ReadData	103
8.7.4.1.2	ReadDataLength	104
8.7.4.1.3	WriteData	104
8.7.4.1.4	ConditionCheckRead	105
8.7.4.1.5	ConditionCheckWrite	106
8.7.4.1.6	Routine_Start	106
8.7.4.1.7	Routine_Stop	107
8.7.4.1.8	Routine_RequestResults	107

8.7.4.1.9	ReturnControlToECU	108
8.7.4.1.10	ResetToDefault	108
8.7.4.1.11	FreezeCurrentState	109
8.7.4.1.12	ShortTermAdjustment	110
8.7.4.1.13	GetScalingInformation	110
8.7.4.1.14	EcuReset	111
8.7.4.2	DSD (Diagnostic Service Dispatcher)	111
8.8	Proposed internal interfaces	112
8.8.1	DSL (Diagnostic Session Layer) Functions	112
8.8.1.1	Dcm_SetSecurityLevel	112
8.8.1.2	Dcm_SetSesCtrlType	113
8.8.1.3	Dcm_GetSesTimingValues	113
8.8.1.4	Dcm_PrepareSesTimingValues	114
8.8.1.5	Dcm_SetSesTimingValues	114
8.8.1.6	Dcm_ResponseOnOneEvent	115
8.8.1.7	Dcm_ResponseOnOneDataByPeriodicId	115
8.8.2	Dsp_DcmConfirmation	116
8.8.3	Dcm_SetNegResponse	117
8.8.3.1	Dcm_ProcessingDone	117
8.8.3.1.1	Dsp_Dcm<DiagnosticService>	118
8.8.3.2	Dcm_StartPagedProcessing	118
8.8.3.3	Dcm_ProcessPage	119
8.8.3.3.1	Dsp_DcmUpdatePage	119
8.8.3.3.2	Dsp_DcmCancelPagedBufferProcessing	120
8.8.4	Dcm_DcmDiagnosticSessionControl	120
8.8.5	Dcm_DcmTesterPresent	120
8.8.6	Dcm_DcmSecurityAccess	121
8.9	DSP (Diagnostic Service Processing)	121
8.9.1	Interface DSP – DEM	121
8.10	AUTOSAR Services implemented by DCM	121
8.10.1	Scope of this chapter	121
8.10.2	Specification of the Ports and Port Interfaces for Diagnostic Services	121
8.10.2.1	General Approach	122
8.10.2.2	Data Types	122
8.10.2.3	Port Interface of Diagnostic Services	124
8.10.3	Definition of the Service DCM	128
9	Sequence diagrams	129
9.1	Overview	129
9.2	DSL (Diagnostic Session Layer)	130
9.2.1	Start Protocol	130
9.2.2	Process Busy behavior	131
9.2.3	Update Diagnostic Session Control when timeout occurs	132
9.2.4	Process single response of ReadDataByPeriodicIdentifier	133
9.2.5	Process single event-triggered response of ResponseOnEvent	135
9.2.6	Process concurrent requests	136
9.2.7	Interface to ComManager	139
9.2.7.1	Handling in Default Session	139
9.2.7.2	Handling in Non-Default Session	140
9.2.7.3	Session transitions	141

- 9.2.7.4 Communication States 142
- 9.3 DSD (Diagnostic Service Dispatcher)..... 143
 - 9.3.1 Receive a request message and transmit a positive response message – synchronous transmission 143
 - 9.3.2 Receive a request message and transmit a positive response message – asynchronous transmission 144
 - 9.3.3 Receive a request message and suppression a positive response 144
 - 9.3.4 Receive request message and transmit negative response message.. 146
 - 9.3.5 Process Service Request with PagedBuffer 147
- 9.4 DSP (Diagnostic Service Processing) 151
 - 9.4.1 Interface DSP – DEM (service 0x19, 0x14, 0x85)..... 151
 - 9.4.2 Interface special services 151
 - 9.4.2.1 Process Diagnostic Session Control..... 151
 - 9.4.2.2 Process Tester Present 153
 - 9.4.2.3 Process Security Access 155
- 10 Configuration specification 156
 - 10.1 How to read this chapter 156
 - 10.1.1 Configuration and configuration parameters 156
 - 10.1.2 Variants 157
 - 10.1.3 Containers 157
 - 10.2 DCM global Configurations 158
 - 10.2.1 Dcm 158
 - 10.2.1.1 COMPONENT_WIDE_PARAMETERS 158
 - 10.2.1.2 PAGE_BUFFER_CFG..... 159
 - 10.3 DSD (Diagnostic Service Dispatcher)..... 160
 - 10.3.1 DcmDsdServiceTable 160
 - 10.3.2 DcmDsdService 161
 - 10.3.3 DcmDsd..... 163
 - 10.4 DSL (Diagnostic Session Layer) 163
 - 10.4.1 DcmDslBuffer..... 164
 - 10.4.2 DcmDslConnection 164
 - 10.4.3 DcmPageBufferCfg..... 165
 - 10.4.4 DcmDslMainConnection 166
 - 10.4.5 DcmDslPeriodicTransmission 167
 - 10.4.6 DcmDslProtocol 167
 - 10.4.7 DcmDslProtocolRow..... 167
 - 10.4.8 DcmDslProtocolRx..... 170
 - 10.4.9 DcmDslProtocolTiming 171
 - 10.4.10 DcmDslProtocolTimingRow 171
 - 10.4.11 DcmDslProtocolTx..... 173
 - 10.4.12 DcmDslResponseOnEvent..... 173
 - 10.5 DSP (Diagnostic Service Processor)..... 174
 - 10.5.1 174
 - 10.5.2 DcmDspReadDTC 174
 - 10.5.3 DcmDspReadDTCRow 174
 - 10.5.4 DcmDspSecurity 176
 - 10.5.5 DcmDspSecurityRow 176
 - 10.5.6 DcmDspSession 178
 - 10.5.7 DcmDspSessionRow 178

10.5.8 DcmGeneral	180
10.5.9 DcmDslDiagResp	181
10.6 Published Information.....	183
11 Changes to Release 1	184
11.1 Deleted SWS Items	184
11.2 Replaced SWS Items	184
11.3 Changed SWS Items.....	184
11.4 Added SWS Items	184

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module DCM.

The Diagnostic Communication Manager (DCM) provides a common API for diagnostic services in the AUTOSAR-Basic-SW. The functionality of this AUTOSAR-Basic-SW is used by external diagnostic tools e.g. in the development, manufacturing or service.

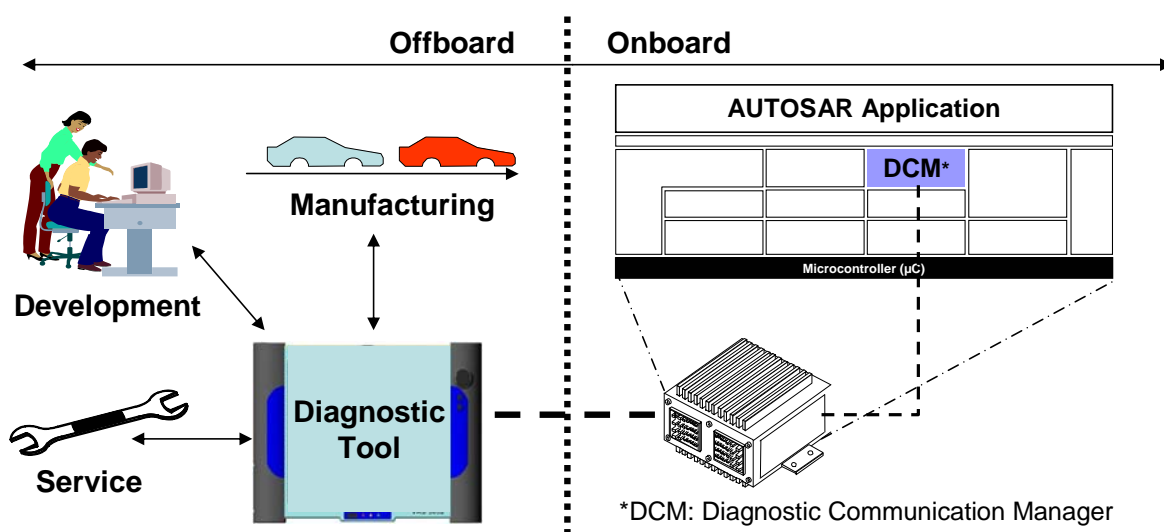


Figure 1 Communication overview between external diagnostic tools and AUTOSAR-SW

The DCM handles different diagnostic protocols.

Dcm019:

One protocol is used for legislated OBD (ISO 15031-5).

Dcm018:

Another protocol is used for enhanced diagnostics (ISO 14229-1).

For diagnostic session handling the network independent sections of the following specifications are also handled by the DCM.

ISO 15765-3: Implementation of unified diagnostic services (UDS on CAN)

ISO 15765-4: Requirements for emissions-related systems

The network dependencies for all networks e.g. CAN, LIN, FlexRay or MOST have to be implemented outside the DCM module. That means the module AUTOSAR PDU Router needs to provide network independency. Only the OSI-Layers 5 to 7 are part of the DCM (see Table 1 Diagnostic protocols and OSI –Layer).

OSI-Layer	Protocols:				
7	UDS-Protocol - ISO14229-1				Legislated OBD - ISO 15031-5
6	-	-	-	-	-
5	ISO15765-3	-	-	-	ISO 15765-4
4	ISO15765-2	-	-	-	-
3	ISO15765-2	-	-	-	ISO 15765-4
2	CAN-Protocol	LIN-Protocol	Flexray	MOST	ISO 15765-4
1	CAN-Protocol	LIN-Protocol	Flexray	MOST	ISO 15765-4

Table 1 Diagnostic protocols and OSI –Layer

The DCM ensures diagnostic data flow and manages the diagnostic states especially diagnostic sessions and security states. Furthermore the DCM checks if the diagnostic service request is supported and if the service may be executed in the current session according to the diagnostic states.

The DCM provides for all diagnostic services (full-set of ISO 14229-1 and ISO 15031-5) interfaces to the AUTOSAR-RTE. Additionally the DCM processes some basic diagnostic services (respectively subservices) for detail information see chapter 7.2.3.

1.1 DCM in AUTOSAR architecture

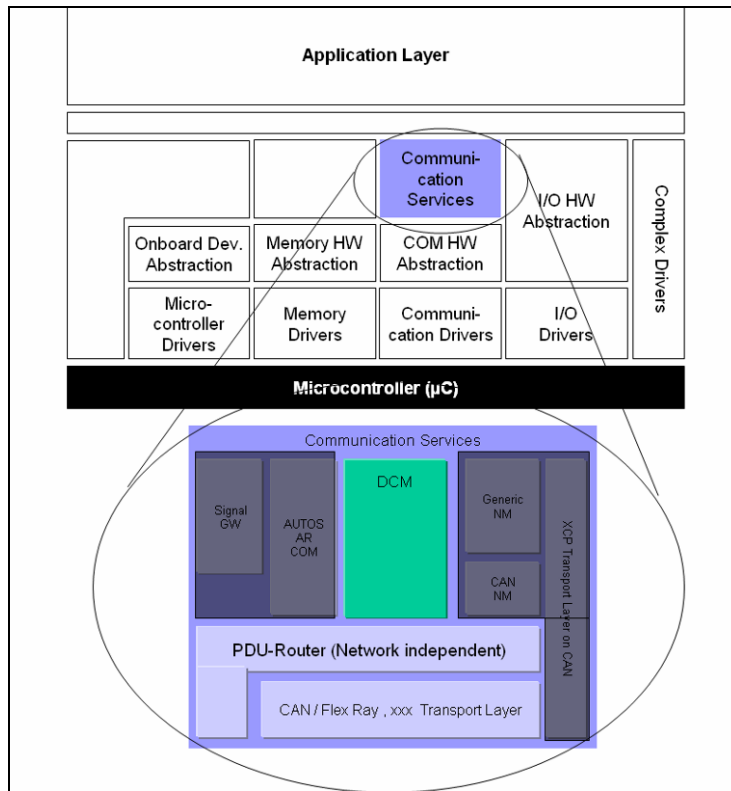


Figure 2 Position of DCM in AUTOSAR Architecture

In the AUTOSAR Architecture the Diagnostic Communication Manager (DCM) is located in the Communication Services (Service Layer).

In the Communication process the DCM receives a Diagnostic message from the PDU Router.

DCM internally the Diagnostic message will be processed, checked and handed on to the Autosar SW Components for further processing. Depending on the Diagnostic service Id the corresponding calls in the Application Layer will be done.

The DCM shall be network independent. This requires a network independent interface to the PDU Router (which handles the networks CAN, LIN, FlexRay and MOST).

2 Acronyms and abbreviations

Abbreviation:	Description:
CAN	Controller Area Network
DCM	Diagnostic Communication Manager
DCM PDU ID	Unique DCM PDU Identifier
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DSD	Diagnostic Service Dispatcher (Part of the DCM)
DSL	Diagnostic Session Layer (Part of the DCM)
DSP	Diagnostic Service Processing (Part of the DCM)
DTC	Diagnostic Trouble Codes
LIN	Local Interconnect Network
MOST	Media Orientated System Transport
NRC	Negative Response Code
OBD	On-Board Diagnosis
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PDU ID	PDU Identifier
PID	Parameter Identifier
RTE	Runtime Environment
SAP	Service Access Point
SID	Service Identifier
SW-C	Software-Components
UDS	Unified Diagnostic Services
Xxx_	Placeholder for an API provider

Definitions:	Description:
Application	An application is equal to an AUTOSAR SW-Component and placed above the RTE.
Application Layer	The application layer is placed above the RTE. Within the application layer the AUTOSAR Software-Components are placed.
Channel	A link on which a data transfer could take place. If there is more then one channel, there is normally some kind of ID connected to the channel.
Diagnostic Channel	A link on which a data transfer between a diagnostic tool and an ECU could take place. Example: An ECU is connected via CAN, so the diagnostic channel is a CAN-ID. Connections to other bus-systems as e.g. MOST, Flexray, LIN , and so on, are also possible.
external diagnostic tool	<p>It is a device which is NOT permanently connected within the vehicle communication network. This device could be connected to the vehicle for various purposes, as e.g.</p> <ul style="list-style-type: none"> • development • manufacturing • service (garage) <p>Know devices are e.g.</p> <ul style="list-style-type: none"> • a diagnostic tester • an OBD scan tool <p>The external diagnostic tool is to be connected by a mechanic to gather information from "inside" the car.</p>
Freeze Frame	A set of the vehicle/system operation conditions at a specific time.

<p>Functional addressing</p>	<p>Functional addressing is used by the external (or internal) tester if it does not know the physical address of the ECU that shall respond to a service request or if the functionality of the ECU is implemented as a distributed server in several ECUs. When functional addressing is used, the communication is a broadcast communication from the external tester to one or more ECUs (1 to n communication).</p> <p>Use cases are (for example) broadcasting messages as “ECUReset” or “CommunicationControl”</p> <p>OBD communication will always be done in Functional addressing mode.</p>
<p>internal diagnostic tool</p>	<p>It is a device/ECU which is connected within the vehicle communication network. The propose of this device/ECU could be a functionality as e.g.</p> <ul style="list-style-type: none"> • advanced event tracking • advanced analysis's <p>for Service.</p> <p>The behavior of the device/ECU could be the same as if it was an external diagnostic tool.</p> <p>The meaning of ‘internal diagnostic tool’ is NOT that it is included in each ECU as a AUTOSAR SW-Component.</p>
<p>Physical addressing</p>	<p>Physical addressing is used by the external (or internal) tester if it knows the physical address of an ECU that shall respond to a service request. When physical addressing is used, a point to point communication takes place (1 to 1 communication).</p> <p>Use cases are (for example) messages as “ReadDataByIdentifier” or “InputOutputControlByIdentifier”</p>

3 Related documentation

3.1 Input documents

- [1] AUTOSAR General Requirements on Basic Software Modules
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_General.pdf
- [2] Specification of Module PDU Router
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_PDU_Router.pdf
- [3] AUTOSAR Requirements on Basic Software Module Diagnostic
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_Diagnostic.pdf
- [4] Specification of Module Communication Manager
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_ComManager.pdf
- [5] AUTOSAR Layered Software Architecture
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_LayeredSoftwareArchitecture.pdf
- [6] AUTOSAR ECU Configuration Specification
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_ECU_Configuration.pdf
- [7] Specification of Diagnostics Event Manager
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_DEM.pdf
- [8] AUTOSAR Basic Software Module Description Template,
https://svn2.autosar.org/repos2/22_Releases/
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

- [9] D1.5-General Architecture; ITEA/EAST-EEA, Version 1.0; chapter 3, page 72 et seq.
- [10] D2.1-Embedded Basic Software Structure Requirements; ITEA/EAST-EEA, Version 1.0 or higher
- [11] D2.2-Description of existing solutions; ITEA/EAST-EEA, Version 1.0 or higher.

- [12] ISO14229-1 Unified diagnostic services (UDS) - Part 1: Specification and Requirements (ISO DIS 26.05.2004)
[Note: in additional to the DIS version DCM will support the following feature
“handling for service 0x19 subfunction 0x14 reportDTCFaultDetectionCounter”]
- [13] ISO15031-5.4 Communication between vehicle and external equipment for emissions-related diagnostics - Part 5: Emissions-related diagnostic services (2005-01-13)
- [14] ISO15765-3: Diagnostics on controller area network (CAN) - Part 3: Implementation of unified diagnostic services (UDS on CAN) (2004-10-06)
- [15] ISO 15765-4 Diagnostics on controller area network (CAN) - Part 4: Requirements for emissions-related systems (2005 01-04)

4 Constraints and assumptions

4.1 Limitations

Due to the time schedule set up for the specification phase following limitations apply when using the DCM basic software module:

- The DCM does not provide any diagnostic multi-channel capabilities. That means parallel requests of a tester addressed to different independent functionalities can not be processed by a single DCM module. Furthermore the concept currently implemented does not take into account that more than one instance of a DCM module resides in one ECU.
As the legislator requires that emissions-related service requests according to ISO 15031-5 shall be processed prior to any enhanced diagnostic requests the DCM provides a protocol switching mechanism based on protocol prioritization.
Different vehicle manufacturers require parallel diagnostic communication as a key diagnostic feature for manufacturing, thus future implementations shall consider this requirement.
- The DCM provides no mechanism to handle the “Retry mechanism” of the Flexray TP, therefore the following requirement exists:
 - The length parameter (which is the buffer request for transmission) is expected by DCM to be always 0 (zero)
- The DCM needs to check the consistency and integrity of the whole message before any further process, therefore the following requirement exists:
 - The length parameter (which is the buffer request for reception) is expected by DCM to be always 0 (zero)
- Support for EcuReset functionality (as defined in ISO 14229) is limited in Autosar, therefore the service EcuReset will only support the sub function HardReset (through Mcu) in the current Autosar Release.
- No communication control functionalities (as defined in ISO 14229) are specified in Autosar module, therefore the service CommunicationControl will not be managed in the current Release.
- No direct memory access (as defined in ISO 14229) is specified in Autosar modules, therefore the following diagnostic services will not be managed in the current release :
 - ReadMemoryByAdress
 - WriteMemoryByAdress
 - RequestDownload

- RequestUpload
- TransferData
- RequestTransferExit
- DynamicallyDefineDataIdentifier for sub-service 02 (Define by memory address)
- Service AccessTimingParameter is not supported by ISO in CAN and LIN. Also it is not planned to support this service with Flexray. Therefore no support for this service is planned.
- SubFonction *onDTCStatusChange* of Service ResponseOnEvent is not supported in the current release.
- Service SecureDataTransmission is not supported in the current release.
- Service LinkControl is not supported in the current release.
- The Central Diagnostic SWC, as defined in the previous Autosar Release, is no more used in the current Release.

4.2 Applicability to car domains

The basic software module DCM can be used for all car domains.

4.3 Applicability to emission related environments (OBD)

Basically this specification is intended to fulfill the emission related requirements given by legislator. However, the supplier of the emission related system is responsible to fulfill the OBD requirements.

Example: During the integration of the DCM Module within the system, the timing requirements (50ms response time) must be fulfilled.

5 Dependencies to other modules

The AUTOSAR **Diagnostic Communication Manager (DCM)** has interfaces and dependencies to the following Basic software modules and Software Components:

- **Diagnostic Event Manager (DEM)**
The DEM shall provide function calls to retrieve all information relating to fault memory, so that the DCM is able to respond on tester requests reading data from fault memory.
- **PDU Router**
The PDU Router shall provide function calls to transmit and receive diagnostic data. Due to physical layer limitations generally data segmentation is necessary. Proper operation of the DCM necessitates that the PDU Router interface supports all service primitives defined for the Service Access Point (SAP) between diagnostic application layer and underlying transport layer. (pls. refer to ISO 14229-1 chapter 5. Application layer services for details).
- **ComManager**
The ComManager shall provide function where the DCM can indicate the states “active” and “inactive” for diagnostic communication. The DCM shall provide functionality to handle the communication requirements “Full- / Silent- / No-Communication”. Additionally the DCM shall guarantee the functionality to enable and disable Diagnostic Communication if requested by the ComManager.
- **SW-Components (SW-C)/ RTE**
The DCM has the capability to analyse the received diagnostic request data stream. All functionalities related to diagnostic communication, as protocol handling, timing and segmented responses shall be done by the DCM. Furthermore the DCM shall handle all diagnostic services internally. If the requested action or information is not provided by the DCM or any other BSW module an adequate port-interface shall be provided to gather the requested information or to perform the requested action.

The figures below depict the placement of the DCM within the AUTOSAR software structure:

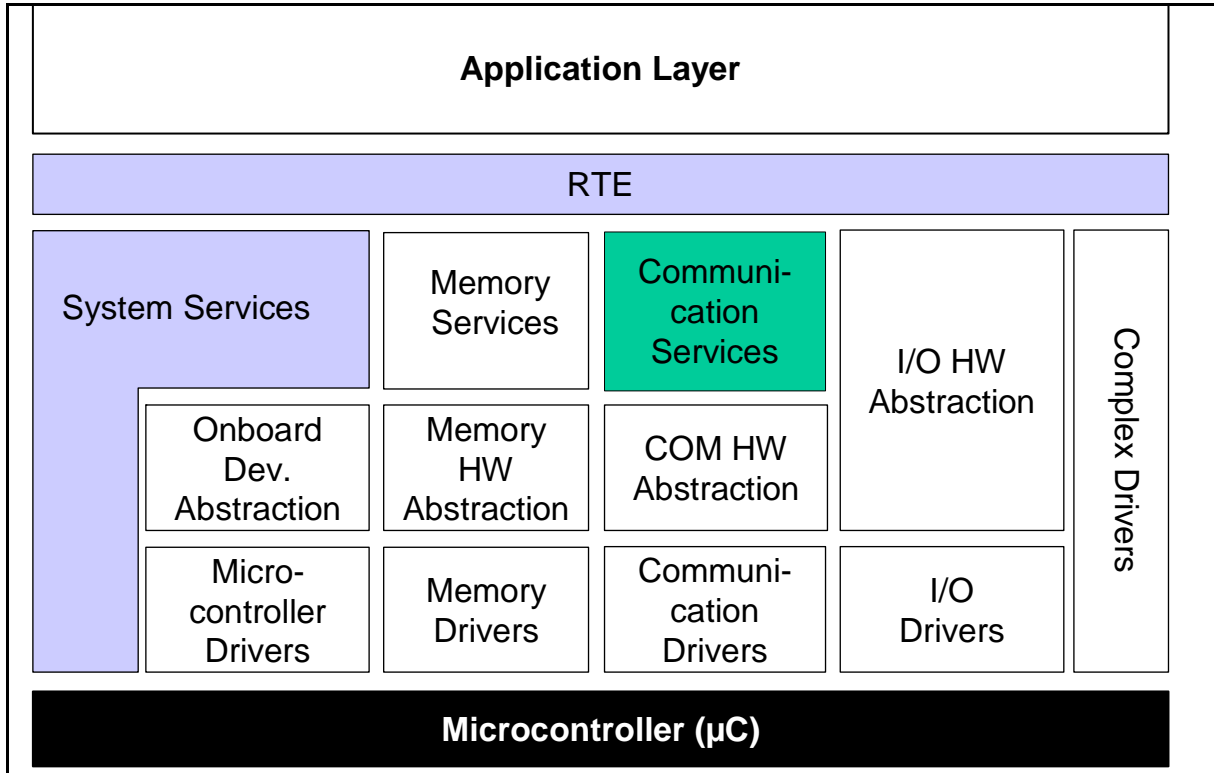


Figure 3: Overview of the DCM interfaces

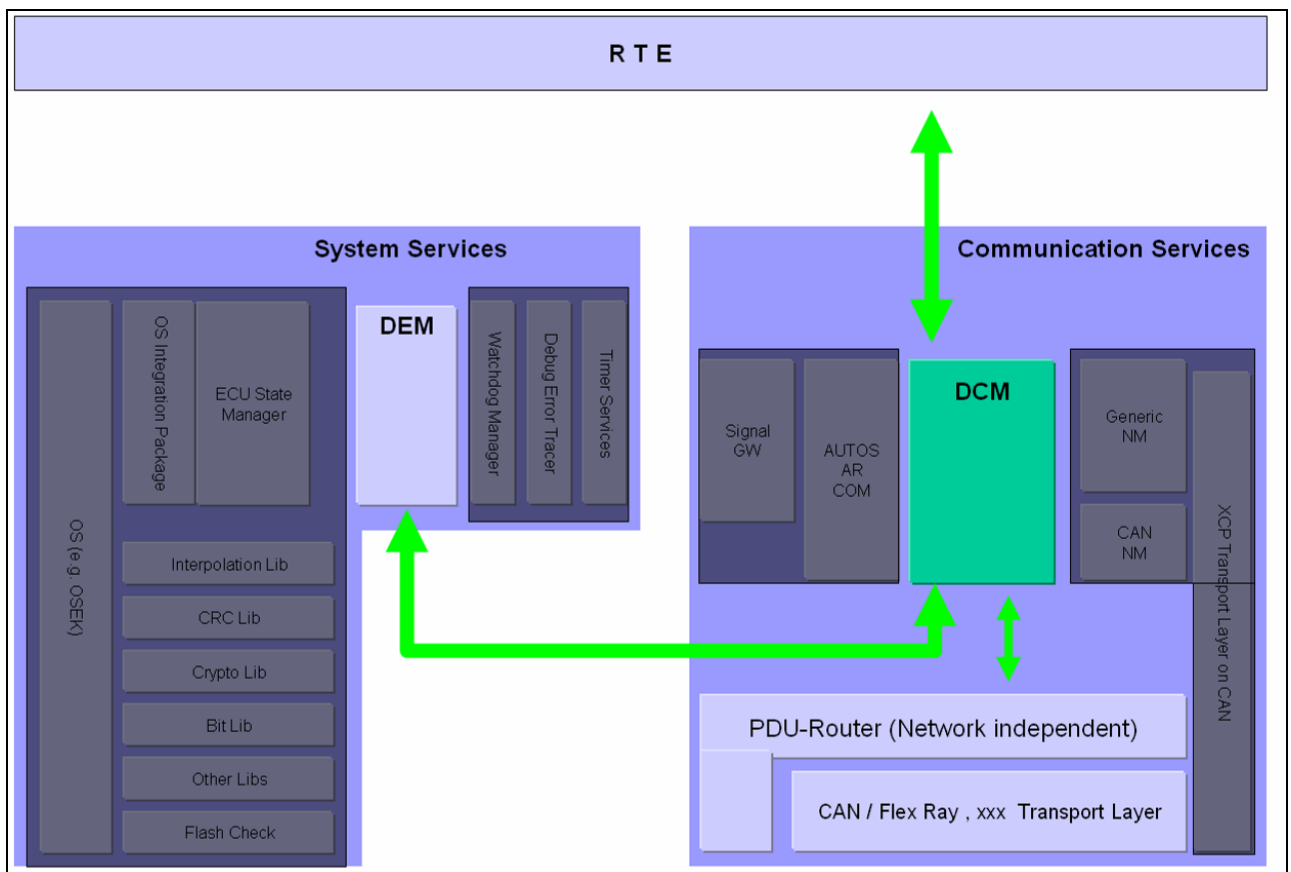


Figure 4: DCM interfaces

5.1 File structure

5.1.1 Code file structure

Dcm054: The code file structure shall not be defined within this specification completely.

The code-file structure shall include the following files named:

- Dcm_Lcfg.c – for link time configurable parameters and
- Dcm_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.

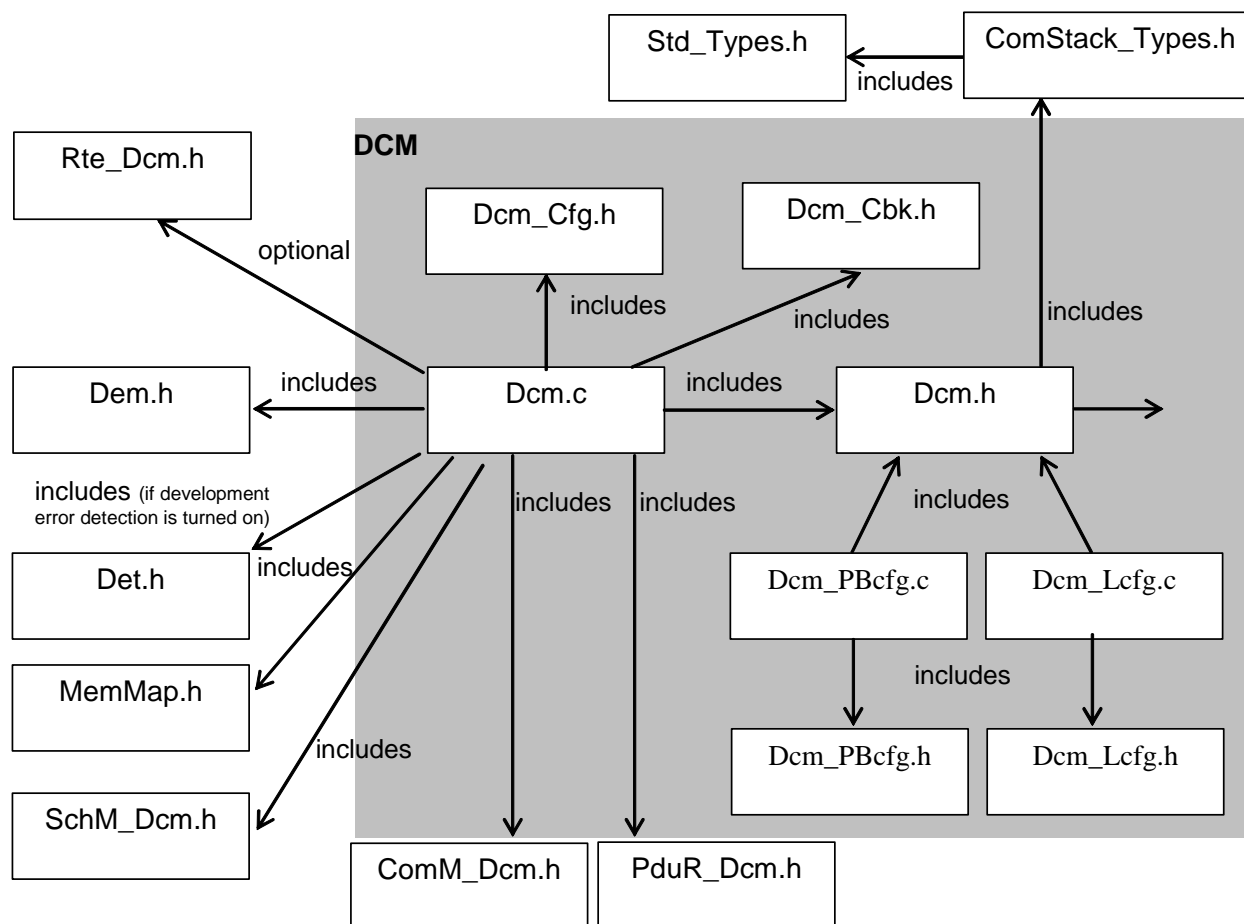


Figure 5: DCM Component file structure

5.1.2 Header file structure

Dcm055:

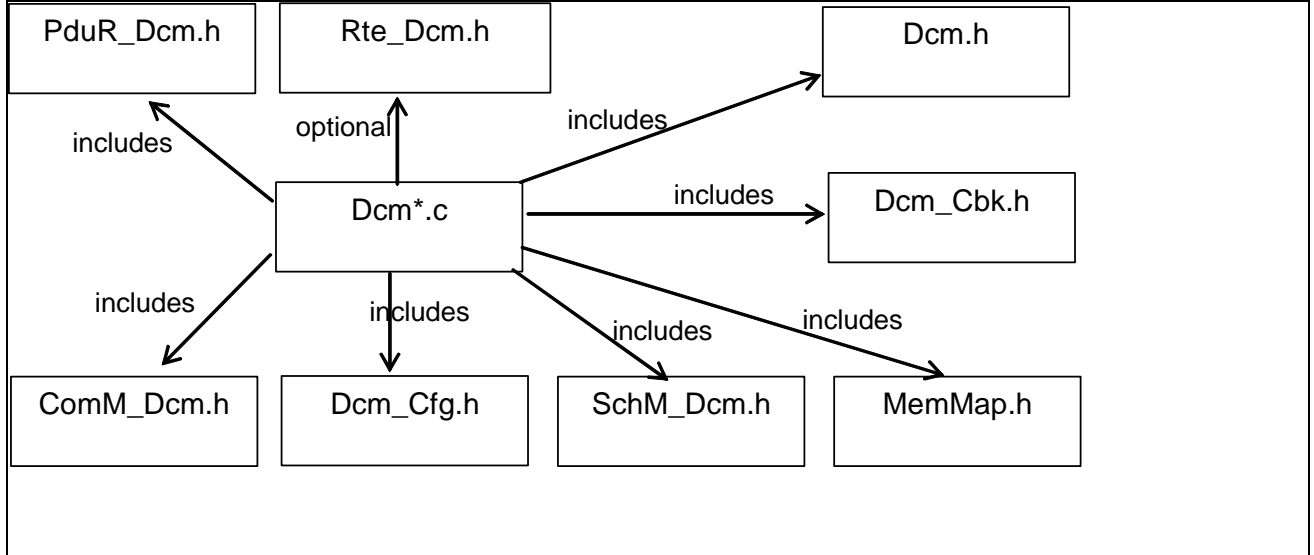


Figure 6: DCM Component header file structure

The public interface of the DCM component is part of the file Dcm.h. This file contains all types, functions and parameters which are visible to any SWC. The implementation is contained in one or multiple Dcm*.c files. Any callbacks supplied by the application are located in Rte_Dcm.h.

As shown in Figure 6 the DCM header files are linked as followed:

- Dcm.c shall include Dcm.h and SchM_Dcm.h

6 Requirements traceability

Document: General Requirements on Basic Software Modules

Functional General Requirements	
Requirement	Satisfied by
Configuration	--
[BSW00344] Reference to link-time configuration	Dcm011
[BSW00404] Reference to post build time configuration	Dcm011
[BSW00405] Reference to multiple configuration sets	Dcm011
[BSW00345] Pre-compile-time configuration	Dcm011
[BSW159] Tool-based configuration	Not applicable
[BSW167] Static configuration checking	Requirement on configuration tool
[BSW171] Configurability of optional functionality	Dcm011
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable
[BSW00380] Separate C-Files for configuration parameters	Dcm054
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Dcm054
[BSW00381] Separate configuration header file for pre-compile time parameters	Dcm055
[BSW00412] Separate H-File for configuration parameters	Dcm055
[BSW00382] Not-used configuration elements need to be listed	Not applicable
[BSW00383] List dependencies of configuration files	Not applicable
[BSW00384] List dependencies to other modules	Fulfilled by chapter 5
[BSW00387] Specify the configuration class of callback function	Not applicable
[BSW00388] Introduce containers	Dcm057
[BSW00389] Containers shall have names	Dcm058
[BSW00390] Parameter content shall be unique within the module	Dcm059
[BSW00391] Parameter shall have unique names	Dcm060
[BSW00392] Parameters shall have a type	Dcm061
[BSW00393] Parameters shall have a range	Dcm062
[BSW00394] Specify the scope of the parameters	Dcm063
[BSW00395] List the required parameters (per parameter)	Dcm064
[BSW00396] Configuration classes	Dcm011
[BSW00397] Pre-compile-time parameters	Dcm064
[BSW00398] Link-time parameters	Dcm064
[BSW00399] Loadable Post-build time parameters	Dcm064
[BSW00400] Selectable Post-build time parameters	Dcm064
[BSW00402] Published information	Dcm013 Dcm075
Wake-Up	--
[BSW00375] Notification of wake-up reason	Not applicable
Initialisation	--
[BSW101] Initialization interface	Dcm033 Dcm034 Dcm035 Dcm036 Dcm037
[BSW00416] Sequence of Initialization	Not applicable
[BSW00406] Check module initialization	Not applicable
Normal Operation	--
[BSW168] Diagnostic Interface of SW components	Not applicable
[BSW00407] Function to read out published parameters	Dcm065

[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable
[BSW00424] BSW main processing function task allocation	Dcm053
[BSW00425] Trigger conditions for schedulable objects	Not applicable
[BSW00426] Exclusive areas in BSW modules	Not applicable
[BSW00427] ISR description for BSW modules	Not applicable
[BSW00428] Execution order dependencies of main processing functions	Not applicable
[BSW00429] Restricted BSW OS functionality access	Not applicable
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path [approved]	Not applicable
[BSW00433] Calling of main processing functions	Not applicable
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable
Shutdown Operation	--
[BSW00336] Shutdown interface	Not applicable
Fault Operation and Error Detection	--
[BSW00337] Classification of errors	Dcm012 Dcm041
[BSW00338] Detection and Reporting of development errors	Dcm040 Dcm042
[BSW00369] Do not return development error codes via API	Dcm044
[BSW00339] Reporting of production relevant error status	Not applicable
[BSW00421] Reporting of production relevant error events	Not applicable
[BSW00422] Debouncing of production relevant error status	Not applicable
[BSW00420] Production relevant error event rate detection	Not applicable
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable
[BSW00323] API parameter checking	Dcm043
[BSW004] Version check	Dcm013
[BSW00435] Header File Structure for the Basic Software Scheduler	Dcm055
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	Dcm055
[BSW00409] Header files for production code error IDs	Not applicable
[BSW00385] List possible error notifications	Not applicable
[BSW00386] Configuration for detecting an error	Not applicable
Non-functional Requirements	--
Software Architecture Requirements	--
[BSW161] Microcontroller abstraction	Not applicable
[BSW162] ECU layout abstraction	Not applicable
[BSW00324] Do not use HIS I/O Library	Not applicable
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable
[BSW00415] User dependent include files	Not applicable
Software Integration Requirements	--
[BSW164] Implementation of interrupt service routines	Not applicable
[BSW00325] Runtime of interrupt service routines	Implementation specific
[BSW00326] Transition from ISRs to OS tasks	Not applicable
[BSW00342] Usage of source code and object code	Not applicable
[BSW00343] Specification and configuration of time	Dcm067
[BSW160] Human-readable configuration data	Fulfilled by chapter DCM in the ECUC SWS
Software Module Design Requirements	--
Software quality	--
[BSW007] HIS MISRA C	Implementation specific
Naming conventions	--

[BSW00373] Main processing function naming convention [approved]	Dcm053
[BSW00350] Development error detection keyword [approved]	Dcm076

Document: AUTOSAR requirements on Basic Software, cluster Diagnostic

Requirements on Basic Software Module Diagnostic	
Requirement	Satisfied by
General	--
[BSW04010] Interface between Diagnostic service handling and Diagnostic event (error) management	Dcm002
Diagnostic communication management (DCM)	--
[BSW04007] Provide Diagnostic service handling	Dcm010
[BSW04021] Switch diagnostic communication access	Dcm015 Dcm016
[BSW04032] Support of different diagnostic addresses	See Release Note
[BSW04080] Support multi-channel capability for diagnostic communication	See Release Note
[BSW04062] Ensure timing during parallel diagnostic requests	Dcm003
[BSW04065] Clearing of events or event groups	Dcm005
[BSW04058] Support individual deletion and reading services for 'Secondary Event Memory'	Dcm007 Dcm008
[BSW04067] Counting and evaluation of events according to ISO 14229-1 DTCStatusMask	Dcm007 Dcm008 Dcm009
[BSW04000] Support Diagnostic Standard UDS (ISO14229-1)	Dcm018 Dcm069 Dcm070 Dcm071 Dcm072 Dcm073 Dcm074
[BSW04001] Support Diagnostic Standard OBD (ISO15031-5)	Dcm019
[BSW04005] SecurityAccess level handling is managed by DCM	Dcm020 Dcm021 Dcm073
[BSW04006] Session handling is managed by DCM	Dcm022 Dcm023
[BSW04016] Provision of Busy Handling [Approved]	Dcm024
[BSW04019] Application callback after transmit confirmation	Dcm045
[BSW04020] Suppression of Responses	Dcm001 Dcm017
[BSW04033] Upload/Download services for data handling	See Release Note
[BSW04036] Format checking of diagnostic services	Dcm026 Dcm071 Dcm074
Timing Requirements	--
[BSW04015] Provision of timing handling according to ISO15765-3	Dcm027
Resource Usage	--
[BSW04017] Provide optimized buffer handling	Dcm028 Dcm038 Dcm068
Interface and API	--
[BSW04078] Interface to fault memory, fault status	Dcm029
[BSW04011] Provide diagnostic state information	Dcm020 Dcm022 Dcm072
[BSW04003] Interface to PDU Router shall be network independent	Dcm030
[BSW04079] The size of a FreezeFrame shall be reported to the DCM by the DEM	Dcm039
Configuration	--
[BSW04059] Configuration of timing parameter	Dcm031
[BSW04024] Configurable size of transferred data	Dcm032

7 Functional specification

7.1 Overview

The **Diagnostic Communication Manager (DCM)** is an intermediate layer between the application and the vehicle network communication (CAN, LIN, FlexRay and MOST) which is capsulated by the PDU Router.

The DCM has an interface with the PDU Router. The DCM itself is network independent. So the protocol and message configuration will be done in layers below the DCM.

DCM has a interface to the 'AUTOSAR Software components' (via RTE).

The DCM consists of following functional blocks:

- DSP – Diagnostic Service Processing
- DSD – Diagnostic Service Dispatcher
- DSL – Diagnostic Session Layer

Overview description for the different functional blocks:

DSP – Diagnostic Service Processing

DSP - Process specific Service (respectively Sub Service) Requests

The DSP is mainly a container for completely implemented diagnostic services that are common amongst the different applications (e.g. access to the fault data) and thus do not need to be implemented by the application.

DSD – Diagnostic Service Dispatcher

The purpose of the DSD (Diagnostic Service Dispatcher) is to process a stream of diagnostic data. Processing in this context means:

Receive a new diagnostic request over a network and forward to a data processor.

Transmit a diagnostic response over a network when triggered by the data processor. (e.g.: DSP).

DSL – Diagnostic Session Layer

The Diagnostic Session Layer (DSL) ensures data flow concerning diagnostic requests and responses. DSL also supervises and guarantees diagnostic protocol timing. Furthermore DSL manages diagnostic states (esp. diagnostic session and security).

7.1.1 DCM Architecture overview

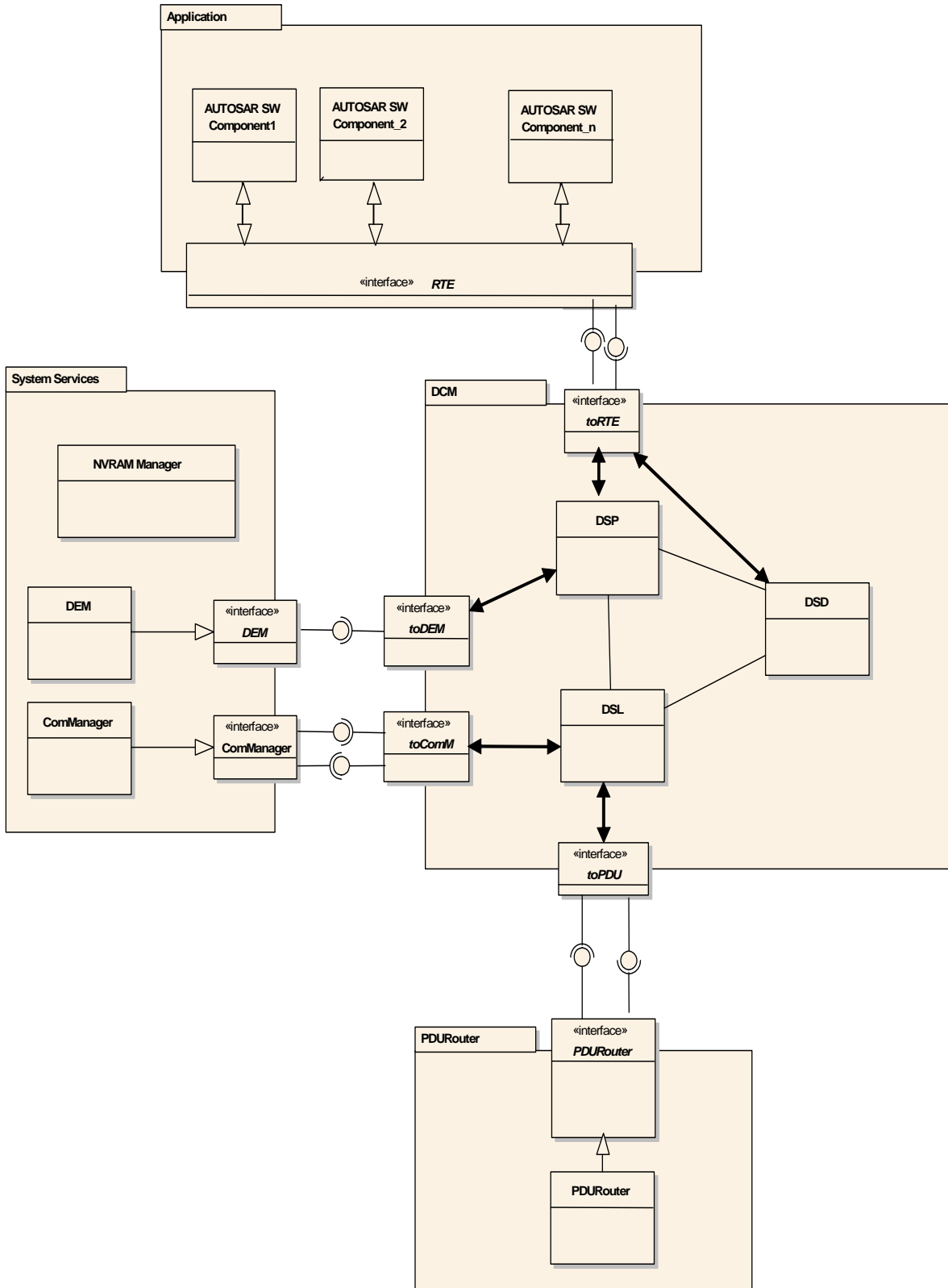


Figure 7: DCM Architecture overview with interfaces

7.2 Functional blocks

7.2.1 DSL (Diagnostic Session Layer)

7.2.1.1 Introduction

The Diagnostic Session Layer (DSL) ensures data flow concerning diagnostic requests and responses. DSL also supervises and guarantees diagnostic protocol timing. Furthermore DSL manages diagnostic states (esp. diagnostic session and security) and communications states required by the Communication Manager.

Dcm030:

All functional areas of DSL are in conformance with the specifications ISO14229-1 (Unified diagnostic services (UDS)) and the network independent part of ISO15765-3 (UDS on CAN). There is no network dependent functional area in the DSL. Within the configuration, some parameters can be set dependent on the network (e.g. the session timing parameter values, see chapter 10.4 DSL (Diagnostic Session Layer).

7.2.1.2 Use cases

Following use cases can be identified:

- Session handling (as required by ISO 14229-1 / ISO 15765-3)
- Application layer timing handling (as required by ISO 14229-1 / ISO 15765-3)
- Specific response behavior (as required in ISO 14229-1 / ISO 15765-3)

All these functionalities are provided by the DSL.

7.2.1.3 Interaction with other modules

Following interaction with other modules exists:

- **PDU Router**
 - o PDU Router provides data of incoming diagnostic requests
 - o DSL triggers output of diagnostic responses
- **DSD (Diagnostic Service Dispatcher)**
 - o DSL informs DSD about incoming requests and provides the data
 - o DSD triggers output of diagnostic responses
- **Software Components / DSP (Diagnostic Service Processing)**
 - o DSL provides access to security and session state
- **Communication Manager**
 - o DSL guarantees the communication behavior required by the Communication Manager

7.2.1.4 Functional Description

7.2.1.4.1 Overview:

Following tasks can be identified for the DSL:

Request Handling

- Forward requests from PDU Router to DSD
- concurrent "TesterPresent" ("keep alive logic")

Response Handling

- Forward responses from DSD to PDU Router
- Guarantee response timing to tester
- Support of periodic transmission
- Support of ResponseOnEvent (ROE) transmission
- Support of segmented response
- Support of ResponsePending response triggered by application

Security Level Handling

- Manage security level

Session State Handling

- Manage session state
- Keep track of active non-default sessions
- Informs depending modules on session change
- Allow to modify timings

Diagnostic Protocol Handling

- Handling of different diagnostic protocols
- Manage resources

Communication Mode Handling

- Handling of communication requirements (Full- / Silent- / No Communication)
- Indicating of active / inactive diagnostic
- Enable / Disable all kinds of diagnostic transmissions

7.2.1.4.2 Detailed description

7.2.1.4.2.1 Forward requests from PDU Router to DSD

The PDU Router indicates to DCM whenever reception of new diagnostic request content is started on a DcmRxDpuld, which is assigned to DCM. This is done with requesting DCM to provide the receive buffer (**Dcm_ProvideRxBuffer()**).

Within this API, the lower layer inform the DCM about the overall number of bytes to be received.

There will be only one call of Dcm_ProvideRxBuffer() per each diagnostic request. So the receive buffer needs to be as large as the maximum diagnostic request length.

DCM can now

- provide a buffer -> return value: BUFREQ_OK
- provide no buffer, since buffer temporarily not available (previous request not finished) -> return value: BUFREQ_E_BUSY
- provide no buffer, since available buffer is smaller than the indicated size of tester request -> return value: BUFREQ_E_OVFL

If reception of diagnostic request is finished (successful or with errors), the PDU Router gives an receive indication to DCM (**Dcm_RxIndication()**).

DSL will forward received data to DSD only after successful receive indication.

More description to the APIs (prototype, input/output parameter) can be found in the interface description of PDU Router (see [1]) and in chapter 8.5.1 DSL (Diagnostic Session Layer) Functions

It is allowed to have different DcmPduld for different diagnostic communication application.

Example:

- OBD DcmRxPduld: for reception of OBD requests
- OBD DcmTxPduld for transmission of OBD responses
- UDS phys DcmRxPduld for reception of UDS physical addressed requests
- UDS func DcmRxPduld for reception of UDS functional addressed request
- UDS DcmTxPduld for transmission of UDS responses

Address Type (physical / functional addressing) is configured per DcmRxPduld (see chapter 10.4 DSL (Diagnostic Session Layer)). A configuration per DcmRxPduld is possible, since there will be always different DcmRxPduld values for functional and physical reception independent of the addressing format of Transport Layer (extended addressing, normal addressing).

7.2.1.4.2.2 Concurrent “TesterPresent” (“keep alive logic”)

It is possible, that functional “TesterPresent” commands are sent by the tester in parallel to physical requests / responses. This is called “keep alive logic” in the ISO14229-1.

This functional “TesterPresent” will be received on a separate DcmRxPduld (UDS func DcmRxPduld) with a separate receive buffer.

Due to that reason, the functional TesterPresent (and only functional TesterPresent without response) is handled in the following way:

When PDU router indicates functional request, DSL checks the content of this request. If the content is equal to “TesterPresent” command with “suppressPosRspMsgIndicationBit” set to TRUE (SID equal to 0x3E, subfunction equal to 0x80), DSL resets the session timeout timer (S3_{Server}). This request will not be forwarded to DSD for further interpretation.

Because of bypassing of functional “TesterPresent” in the DSL, the DCM is able to receive and process next physical request without delay.

7.2.1.4.2.3 Forward responses from DSD to PDU Router

The DSD will request the DSL for transmission of responses.

After minimum response time is over ($P2_{ServerMin}$), DSL will trigger the transmit to the PDU Router. This is done by using the function **PduR_DcmTransmit()**. With this API only the length information is given to PDU Router.

Responses are sent with this $DcmTxPduId$, which is linked in the DCM configuration to the $DcmRxPduId$, the request was received (more details can be found in protocol configuration in chapter 10.4 DSL (Diagnostic Session Layer)).

When **PduR_DcmTransmit()** is called, lower layers will request DCM to provide the transmit buffer / data. This is done with requesting **Dcm_ProvideTxBuffer()**.

The DSL will receive a confirmation after the complete DCM PDU has successfully been transmitted or an error occurred (**Dcm_TxConfirmation()**). This confirmation will be forwarded to the DSD.

In case of a failed transmission (failed **PduR_DcmTransmit()** request or error confirmation (= **Dcm_TxConfirmation()** with error)), the response will not be repeated.

More description to the APIs (prototype, input/output parameter) can be found in the interface description of PDU Router (see [1])

7.2.1.4.2.4 Guarantee timing to tester by sending busy responses

Dcm024:

If the Application (or the DSP) is able to perform the diagnostic task, but needs additional time to finish the task and prepare the response, the DSL needs to guarantee the response timing to tester. This is done by sending a negative response with response code 0x78 (requestCorrectlyReceived-ResponsePending) before reaching the response time ($P2_{ServerMax}$ respectively $P2^*_{ServerMax}$).

This response needs to be sent from a separate buffer, in order to avoid overwriting the ongoing processing of request (e.g. application already prepared response contents in the diagnostic buffer).

The number of negative responses with response code 0x78 for one diagnostic request is limited: **DcmDsIDiagRespMaxNumRespPend**. This avoids deadlock in application.

In case this limit value is reached, DCM will stop processing the active diagnostic request and send a negative response with response code 0x10 (generalReject).

Sequence diagram is available in chapter 9.2.2 Process Busy behavior

Referenced configuration parameters can be found in chapter 10.4 DSL (Diagnostic Session Layer)

7.2.1.4.2.5 Support of periodic transmission

The UDS service “ReadDataByPeriodicIdentifier” (SID 0x2A) allows the tester to request the periodic transmission of data record values from the ECU identified by one or more periodicDataIdentifiers.

TYPE1 = messages on the DcmTxPduId already used for normal diagnostic responses. The outgoing messages must be synchronized with “normal outgoing messages”, which have a higher priority.

TYPE2 = messages on a separate DcmTxPduId.

This type information shall be configured in **DcmDslProtocolTransType**.

In case of TYPE2, the separate PduId shall be configured with **DcmDslPeriodicTxPduRef**.

Responses generated by periodic transmission are sent with a separate protocol and so a separate buffer of configurable size.

The **RefConnForPeriodic** configuration parameter allows linking the protocol used to receive the periodic transmission request to the protocol used for the response.

Note that multiple TxPduIds can be assigned to the periodic transmission protocol

According to the communication mode (see chapter 7.2.1.4.2.16 Communication Mode Handling) the following restrictions shall be respected:

- Periodic transmission communication only in Full Communication Mode
- Periodic transmission events occurred when not in Full Communication Mode
- Periodic transmission events beside Full Communication Mode shall be discarded and not queued for transmission
- Periodic transmission events requested by the application shall not activate the Full Communication Mode

A sequence diagram including a description is available in chapter 9.2.4 Process single response of ReadDataByPeriodicIdentifier

Referenced configuration parameters can be found in chapter 10.4 DSL (Diagnostic Session Layer).

7.2.1.4.2.6 Support of ROE transmission

With the UDS service “ResponseOnEvent” (SID 0x86), a tester requests an ECU to start or stop transmission of responses initiated by a specified event.

Upon registering an event for transmission, the tester also specifies the corresponding service to respond to (e. g. ReadDataByIdentifier).

TYPE1 = Responses are sent on the same DcmTxPduId which was used for the ROE service response. Therefore the DSP has to store the DcmRxPduId which is provided in the **pMsgContext** parameter of the ROE service function. This stored DcmRxPduId has to be forwarded as parameter in the **Dcm_ResponseOnOneEvent** function, where it is used to simulate a request. Those event-triggered responses must be synchronized with “normal” responses, which have a higher priority.

TYPE2 = Responses are sent on a separate DcmTxPduId.

This type information shall be configured in ***DcmDslProtocolTransType***.
In case of TYPE2, the separate channel shall be configured with ***DcmDslROETxPduRef***

Responses generated by “ResponseOnEvent” use a separate protocol and so a separate buffer of configurable size. The configured protocol buffer is used for transmission of the ROE messages (as the reception used a separate protocol, a separate buffer is used for reception). The content of the ***pMsgContext*** pointer (ROE message) shall be copied into the buffer.

The ***RefConnForROE*** configuration parameter allows linking the protocol used to receive the ROE request to the protocol used for the response.

According to the communication mode (see chapter 7.2.1.4.2.16 Communication Mode Handling) the following restrictions shall be respected:

- ROE communication only in Full Communication Mode
- ROE events shall be disabled in any other Communication Mode except Full Communication Mode
- ROE events beside Full Communication Mode shall be discarded and not queued for later transmission
- ROE events requested by the application shall not activate the Full Communication Mode

Please note the following limitations:

While processing an ROE-Event any addition requests (external and internal) will be rejected with the same or lower priority of it's Protocol Table. This restriction is independent of using other diagnostic request/response CAN identifiers.

A sequence diagram including a description is available in chapter 9.2.5 Process single event-triggered response of ResponseOnEvent

You can find referenced configuration parameters in chapter 10.4 DSL (Diagnostic Session Layer).

7.2.1.4.2.7 Support of segmented response (“PagedBuffer”)

Dcm028: If enabled (configuration parameter ***DcmPagedBufferEnabled*** set to TRUE), DCM provides a mechanism to send responses larger than the configured and allocated diagnostic buffer (configured per protocol, see configuration parameter ***DcmDslProtocolBufferID***).

With “PagedBuffer” handling the ECU is not forced to provide a Buffer, which is as large as the maximum length of response.

A sequence diagram showing the communication flow towards application and PDU Router is available in chapter 9.3.5 Process Service Request with PagedBuffer.

Please note:

- “PagedBuffer” handling is for transmit only / no support for reception

- “PagedBuffer” handling is not available for the application (DCM internal usage only)

7.2.1.4.2.8 Support of ResponsePending response triggered by application

In some case application needs to request an immediate negative response with code 0x78 (requestCorrectlyReceived-ResponsePending), which should be send immediately and not just before reaching the response time ($P2_{ServerMax}$ respectively $P2^*_{ServerMax}$).

One usecase is flashprogramming:

To start the flash programming sequence, a diagnostic tester will give the command DiagnosticSessionControl with DiagnosticSessionType set to "ProgrammingSession". It is now good practice that the ECU first sends a negative response ResponsePending, before jumping to bootloader.

If enabled (DCM_FORCE_RESPPEND_ENABLED parameter), the application (which runs in Dcm_GetSesChgPermission()) can trigger the transmission of negative response ResponsePending request by the API Dcm_ForceRespPend().

This response needs to be sent from a separate buffer, in order to avoid overwriting the ongoing processing of request (e.g. application already prepared response contents in the diagnostic buffer).

7.2.1.4.2.9 Manage security level

Dcm020: The DSL module saves the level of the current active security level. For accessing this level, DSL offers interface to

- get the current active security level: **Dcm_GetSecurityLevel()**
- set a new security level: **Dcm_SetSecurityLevel()**

Dcm033: During ECU initialization the security level is set to the value 0x00. With this the ECU is locked.

Security level is reset to the value 0x00 (means the security is enabled) in the following cases:

- with a transition from any diagnostic session other than the defaultSession to another session other than the defaultSession (including the currently active diagnostic session)
- with a transition from any diagnostic session other than the defaultSession to the defaultSession

(**Dcm_SetSecurityLevel()**) (initiated by service 'DiagnosticSessionControl' or $S3_{Server}$ timeout)

More explanation can be found in [11] chapter 8.2.

Only one security level can be active at a time.

More description is given in the API chapter of DSL (see chapter 8.4.2 DSL (Diagnostic Session Layer) Functions).

7.2.1.4.2.10 Manage session state

Dcm022: DSL module saves the state of the current active session.

For accessing this variable, DSL offers interface to

- get the current active session: **Dcm_GetSesCtrlType()**
- set a new session: **Dcm_SetSesCtrlType()**

Dcm034: During ECU initialization the session state is set to the value 0x01 (“DefaultSession”).

More description is given in the API chapter of DSL (see chapter 8.4.2 DSL (Diagnostic Session Layer) Functions).

7.2.1.4.2.11 Keep track of active non-default sessions

Whenever a non-default session is active, DSL needs to supervise the session timeout time ($S3_{Server}$). When the session timeout is reached without receiving any diagnostic request, DSL resets to the default session state (“DefaultSession”, 0x01).

According to following table, the start / stop of $S3_{Server}$ timeout timer is processed:

Sub-sequent start	Completion of any final response message or an error indication (Dcm_TxConfirmation()): confirmation of complete PDU or indication of an error
	Completion of the requested action in case no response message (positive and negative) is required / allowed.
	Indicates an error during the reception of a multi-frame request message. (Dcm_RxIndication()): indication of an error
Sub-sequent stop	Start of a multi-frame request message (Dcm_ProvideRxBuffer()): indicates start of PDU reception
	Reception of single-frame request message. (Dcm_ProvideRxBuffer()): indicates start of PDU reception

"Start of $S3_{Server}$ " means reset the timer and start counting from the beginning.

Sequence diagram is available in chapter 9.2.3 Update Diagnostic Session Control when timeout occurs.

7.2.1.4.2.12 Informs depending modules on session change

Whenever the value of the active session changes (including “self-transitions”) (initiated by service “DiagnosticSessionControl” or $S3_{Server}$ timeout), DSL informs the application by activation of the following function:
Dcm_SesCtrlChangeIndication().

This function needs to be provided by the application.

More description is given in the API chapter of DSL (see chapter 8.7.3.1 DSL (Diagnostic Session Layer) Functions)

7.2.1.4.2.13 Allow to modify timings

Dcm027: Protocol timing is identified with the following timing parameters:

P2ServerMin, P2ServerMax, P2*ServerMin, P2*ServerMax, S3Server
Definition of these parameters can be found in the ISO15765-3 document.

P2min / P2*min and S3Server will be set to defined values:

The following values will be taken: P2min = 0ms, P2*min = 0ms, S3Server = 5s

All protocols (OBD, enhanced diagnosis) share the same protocol timing values (given by default session configuration). These timing values are set, when the protocol is started (More description to “protocol start” is given in chapter 7.2.1.4.2.14 Handling of different diagnostic protocols).

These protocol timing parameters have influence on the session layer timing (no influence on Transport Layer timing).

Some of these timing parameters can be modified while protocol is active with the following means:

- Service “DiagnosticSessionControl”
- Service “AccessTimingParameter”

DSL provides the following interfaces to modify the timing parameters:

- **Dcm_GetSesTimingValues():** to read active timing parameters
- **Dcm_PrepareSesTimingValues():** to prepare setting of new timing parameters (examination, if new values are within the configured limits (see configuration structure **DcmDslProtocolTimeLimit**) (see chapter 10.4 DSL (Diagnostic Session Layer))
- **Dcm_SetSesTimingValues():** to set the new timing parameters. Activation of new timing values is only allowed after sending the response

More description is given in the API chapter of DSL (see chapter 8.4.2 DSL (Diagnostic Session Layer) Functions)

7.2.1.4.2.14 Handling of different diagnostic protocols

It is necessary to distinguish between different diagnostic protocols (e.g. OBD, enhanced diagnosis ...).

This is required because of

- different protocol settings
- different valid service table
- prioritization / preemption of protocol

Different protocol setting

The different diagnostic protocols may have different protocol settings
Configuration details can be found in chapter 10.4 DSL (Diagnostic Session Layer).

Different service tables

For the different protocols a different set of allowed diagnostic services is valid (e.g. the UDS commands for the enhanced diagnosis, the OBD mode services for the OBD protocol). It is possible to create different service tables and link them to the diagnostic protocol (in the configuration).

Dcm035: With every protocol initialization (see chapter section “detection of protocol start”), DSL sets a link to that service table (see configuration parameter **DcmDslProtocolSIDTable**) referenced in the protocol configuration (see chapter 10.4 DSL (Diagnostic Session Layer)).
DSD is using this link for further processing of diagnostic requests.

Prioritization of protocol

Dcm003: Dcm015: Possible use case:

There are ECUs, communicating with a vehicle internal diagnostic tester (running on enhanced diagnosis) and a vehicle external OBD tester. Requirement is here to give the OBD communication higher priority than the enhanced diagnosis. This can be supported with giving the protocol a priority information (see configuration parameter **DcmDslProtocolPriority**) referenced in the protocol configuration.
A Protocol with higher priority is allowed to preempt the already running protocol.

Differentiation of diagnostic protocols is possible, because of different DcmRxDuld values (configured per protocol, see configuration parameter **DcmDslProtocolRxPduRef**) referenced in the protocol configuration.

Dcm016: In chapter 9.2.6 Process concurrent requests a sequence diagram is available, showing the preemption of an enhanced diagnostic protocol with an OBD request.

Preemption of protocol

If a running diagnostic request is preempted by a higher prior request (of an other protocol, e.g. OBD), application is requested to abort further processing of running request by calling **Dcm_StopProtocol()** and finish with **Dcm_ProcessingDone()**. For further details take a look at the sequence charts 9.2.6 Process concurrent requests.

Dcm079: In order to cancel pending transmission in lower-layer, related to the lower prior request, the DCM shall call the interface **PduR_DcmCancelTransmitRequest()** The PduR will later call the **Dcm_TxConfirmation()** interface for confirming the cancellation.

Detection of protocol start

Dcm036: With first request of a diagnostic protocol (OBD, enhanced diagnosis,..), DSL informs the application with following checkcondition callback function: **Dcm_StartProtocol()** (see chapter 9.2.1 Start Protocol).

Inside this function, application can examine the environment conditions and enable / disable further processing of the protocol.

When application allows start of protocol, the protocol properties are set:

- default timing parameters are loaded from default session configuration (see configuration structure **DcmDspSessionRow**).
- Service table is set (see configuration parameter **DcmDslProtocolSIDTable** referenced in the protocol configuration).

Also the security state and the session state are reset.

At protocol start all the service interpreters which provides service init function per configuration element **DcmDsdSidTabSidInitFunction** gets initialized.

Parallel execution of diagnostic response

Dcm081: The DCM shall allow the parallel execution of protocol response in the case of ResponseOnEvent and PeriodicTransmission services.

Dcm082: The DCM shall provide a configuration parameter **DcmDslProtocollsParallelExecutable** to allow a protocol to be executed in parallel of an other one.

Dcm083: Only protocols used for ResponseOnEvent and PeriodicTransmission services can have the configuration parameter **DcmDslProtocollsParallelExecutable** set to TRUE

7.2.1.4.2.15 Manage resources

Due to limited resources the following points should be considered as hints for the design:

- It is allowed to use and allocate only one diagnostic buffer in the DCM. This buffer is then used for processing the diagnostic requests and responses.
- Output of "Response Pending" responses (NRC 0x78) is done with a separate buffer

- PagedBuffer handling (see chapter 7.2.1.4.2.7 Support of segmented response (“PagedBuffer”))

7.2.1.4.2.16 Communication Mode Handling

No Communication:

With the callback routine `Dcm_ComM_NoComModeEntered()`, called by the Communication Manager, all kind (receive and transmit) of communication shall be disabled. That means that the message reception and also the message transmission shall be off.

With the indication of the No Communication Mode the following functionalities shall be disabled:

- ResponseOnEvent transmissions
- PeriodicId transmissions (`ReadDataByPeriodicIdentifier`)
- Normal transmissions

The function `PduR_DcmTransmit` shall not be called even if the functions `DCM_ResponseOneDataByPeriodicId()` or `DCM_ResponseOnOneEvent()` are called in DCM.

Silent Communication:

With the callback routine `Dcm_ComM_SilentComModeEntered()`, called by the Communication Manager, all kind of outgoing (transmit) communication shall be disabled. That means that the message reception shall be on and the message transmission shall be off.

With the indication of the Silent Communication Mode the following functionalities shall be disabled:

- ResponseOnEvent transmissions
- PeriodicId transmissions (`ReadDataByPeriodicIdentifier`)
- Normal transmissions

The function `PduR_DcmTransmit` shall not be called even if the functions `DCM_ResponseOneDataByPeriodicId()` or `DCM_ResponseOnOneEvent()` are called in DCM.

Full Communication:

With the callback routine `Dcm_ComM_FullComModeEntered()`, called by the Communication Manager, all kind of communication shall be enabled. That means that the message reception and also the message transmission shall be on.

With the indication of the Full Communication Mode the following functionalities shall be enabled:

- ResponseOnEvent transmissions
- PeriodicId transmissions (`ReadDataByPeriodicIdentifier`)
- Normal transmissions

The function `PduR_DcmTransmit` and also the functions `DCM_ResponseOneDataByPeriodicId()` or `DCM_ResponseOnOneEvent()` shall be handled without restrictions given by the communication mode.

Default Session:

With the first request of a diagnostic protocol (OBD, enhanced diagnosis,..), the DSL shall inform the Communication Manager with the following callback function **ComM_DCM_ActiveDiagnostic()** about the need to stay in Full Communication Mode.

With the reception of `Dcm_TxConfirmation` connected to the response given by the DSL, the DSL shall inform the Communication Manager with the following callback function **ComM_DCM_InactiveDiagnostic()** about the fact, that Full Communication is not longer needed.

The command "`ComM_DCM_InactiveDiagnostic()`" shall not be sent for "requestCorrectlyReceived-ResponsePending" (NRC 0x78). The command "`ComM_DCM_InactiveDiagnostic()`" shall be sent with the very last response (positive or negative) connected to the request.

If a „suppressPosRspMsgIndicationBit“ is indicated and the positive Response will be suppressed, the command "`ComM_DCM_InactiveDiagnostic()`" shall be called. (please refer to 9.3.3)

Session Transitions:

If the actual diagnostic session shall be changed with the function **Dcm_SesCtrlChangeIndication()** into a session different than the default session, the Communication Manager shall be informed with the callback function **ComM_DCM_ActiveDiagnostic()** about the need to stay in Full Communication Mode.

If the actual diagnostic session shall be changed with the function **Dcm_SesCtrlChangeIndication()** or with the elapse of the timer "`S3Server`" from a session different than the default into the default session, the Communication Manager shall be informed with callback function **ComM_DCM_InactiveDiagnostic()** about the fact, that Full Communication is not longer needed.

Non Default Session:

As long as the server is in a session other than the default session, the function **ComM_DCM_ActiveDiagnostic()** shall not be called when receiving a request from a client, provided by the PDU Router.

As long as the server is in a session other than the default session, the callback function **ComM_DCM_InactiveDiagnostic()** shall not be send with the reception of `Dcm_TxConfirmation` connected to the response given by the DSL.

7.2.2 DSD (Diagnostic Service Dispatcher)

7.2.2.1 Introduction

DCM010: The purpose of the DSD (Diagnostic Service Dispatcher) is to process a stream of diagnostic data. Processing in this context means:

- Receive a new diagnostic request over a network and forward to a data processor.
- Transmit a diagnostic response over a network when triggered by the application (AUTOSAR SW-Component or DSP).

The DSD is responsible to check the validity of an incoming diagnostic request (Verification of Diagnostic Session / Security Access levels / application permission).

Only valid requests will be processed, invalid ones will be rejected automatically.

The validity of an outgoing response must be ensured by the application.

The DSD keeps track of the progress of a service request execution.

The DSD is independent of any network type.

7.2.2.2 Use cases

The following use cases are relevant:

- Receive a request message and transmit a positive response message
- Receive a request message and suppress a positive response
- Receive a request message and transmit a negative response message
- Send a positive response message without corresponding request
- Segmented Responses

Receive a request message and transmit a positive response message

This is the standard use case of normal communication („ping-pong”).

The server (the ECU) receives a diagnostic request message. The DSD ensures the validity of the request message. In this use case, the request is valid and the response will be positive. The request will be forwarded to the appropriate data processor.

When the data processor has finished all actions of data processing, it triggers the transmission of the response message by the DSD.

If the data processor processes the service immediately as part of the request indication function, the application can trigger the transmission inside this indication function (“synchronous”).

If the processing takes a longer time (e. g. waiting on EEPROM driver), the data processor defers some processing (“asynchronous”). The response pending mechanism is covered by the DSL. Again, the data processor triggers the transmission explicitly, but from within the data processor (AUTOSAR SW-Component or DSP) context.

As soon as a request message is received, the corresponding DcmPduld is blocked by the DSL. During the processing of this request, no other request of the same protocol type (e.g. an enhanced session can be ended by a OBD session) can be

received, until the corresponding response message is sent and the DcmPdulId is released again.

For further details please refer to 9.3 DSD (Diagnostic Service Dispatcher).

Receive a request message and suppress a positive response

This is a sub use-case of the last one.

Within the UDS protocol it is possible to suppress the positive response by setting a special bit in the request message.

To offer a uniform handling to the application, this special suppression handling is completely performed within the DSD.

For further details please refer to 9.3 DSD (Diagnostic Service Dispatcher).

Receive a request message and transmit a negative response message

There are a many different reasons why a request message is rejected and a negative response is to be sent.

If a diagnostic request is not valid or if a request may not be executed in the current session, the DSD will reject the processing and a negative response will be returned. But there are even many reasons to reject the execution of a well-formed request message, e. g. if the ECU or system state does not allow the execution.

In this case, the application will register a negative response including a negative response code supplying additional information why this request was rejected.

In case of a request composed of several parameters (e.g. a ReadDataByIdentifier request with more than one identifier to read), each parameters is treated separately. And each of these parameters can return an error. These kinds of request involve a positive response only if all the parameters were processed successfully.

If at least one of these parameters report an error, the DSD shall transmit a negative response with the first error code reported and no positive response shall be send for this request.

For further details please refer to 9.3 DSD (Diagnostic Service Dispatcher).

Send a positive response message without corresponding request

There are two services within the UDS protocol, where multiple responses are sent for only one request. In general, one service is used to enable (and disable) an event- or time-triggered transmission of another service, which again is sent by the ECU without a corresponding request (for details, please refer to ISO 14229-1).

These services are:

- ReadDataByPeriodicIdentifier (0x2A). This service allows the client to request the periodic transmission of data record values from the server identified by one or more periodicDataIdentifiers.

Type 1 = USDT messages on the DcmTxPdulId already used for normal diagnostic responses (single frames only); Type 2 = UUDT message on a separate DcmTxPdulId. For Type 1, the outgoing messages must be synchronized with “normal outgoing messages”, which have a higher priority.

- ResponseOnEvent (0x86). This service requests a server to start or stop transmission of responses on a specified event.

Way 1 = USDT messages on the DcmTxPduld already used for normal diagnostic responses, Way 2 = USDT messages on separate DcmTxPduld. For Way 1, the outgoing messages must be synchronized with “normal outgoing messages”, which have a higher priority.

This handling is especially controlled by the DSL. However, the DSD shall provide the possibility to generate a response without a corresponding request.

For further details please refer to 9.3 DSD (Diagnostic Service Dispatcher).

Segmented Responses (e.g. Paged buffer)

Within the diagnostic protocol, some services allow to exchange a significant amount of byte data, e. g. ReadDTCInformation (SID 19) and TransferData (SID 36).

In the conventional approach, the ECU internal buffer must be as large to keep the longest data message which is allowed to exchange (worst-case). On the other side, RAM memory in ECU is considered to be a critical resource, especially in smaller micros. However, the complete buffer is filled before the transmission is started.

In a more memory-saving approach, the buffer is filled only partly, transmitted partly and then refilled partly – and so on. This paging mechanism requires only a quite small amount of memory, but it demands a well-defined reaction time for buffer refilling. However, the total amount of memory is reduced significantly.

For further details please refer to 9.3 DSD (Diagnostic Service Dispatcher).

The user can decide whether to use the linear or paged buffer for diagnostics.

7.2.2.3 Interaction with other modules

The DSD (Diagnostic Service Dispatcher) is embedded in the DCM in-between the DCM functionality DSL and the DCM functionality DSP (or AUTOSAR SW-Components).

The DSD is called by the DSL when receiving a diagnostic message delegates processing of request to application, i. e. DSP or SW-Component keeps track of request processing transmits the response of the application to the DSL

The DSD (Diagnostic Service Dispatcher) interacts with following DCM functionality:
 DSL(Diagnostic Session Layer)
 DSP (Diagnostic Service Processing) and AUTOSAR SW-Components (the Application)

Interaction with the DSL main functionality (Diagnostic Session Layer)

<i>Direction</i>	<i>Explanation</i>
Bidirectional	- Exchange of the Diagnostic Messages (receive/transmit)
DSD -> DSL	- Receive of latest diagnostic session and latest security level
DSL -> DSD	- Confirmation of transmission of Diagnostic Message

Table 2 Interaction DSD with the DSL

Interaction with the DSP main functionality (Diagnostic Service Processing) and AUTOSAR SW-Components main functionality (the Application)

Direction	Explanation
DSD -> DSP	- Delegate processing of request - Confirmation of transmission of Diagnostic Message
DSP -> DSD	- Signal, that processing is finished

Table 3 Interaction DSD with the DSP**7.2.2.4 Functional Description**

Main Task of the Diagnostic Service Dispatcher (DSD) is the preprocessing and the transmission of diagnostic data request to the application (i.e. DSP or SW-Component) and as well as the assembling of a positive or negative Response Message.

Triggered by a Data Indication functionality (of the DSL) the preprocessing of an incoming Diagnostic Message is started.

Therefore the included service id of this Request is checked and preprocessed based on information taken from configuration data.

However, this configuration data shall comply with Enhanced Diagnostics:
Road vehicles - Unified diagnostic services (UDS) - Part 1: ISO14229-1 (UDS),
and for emission related systems, OBD/CARB: ISO15031.

To handle the processing the following tasks of the Diagnostic Service Dispatcher (DSD) can be identified:

- Support check of diagnostic service identifier and adaptation of the Diagnostic message
- Handling of „suppressPosRspMsgIndicationBit“
- Verification functionality
- Distribution of Diagnostic Message/Data to data processor (DSP or AUTOSAR Software Component)
- Assemble positive or negative response
- Initiate transmission

7.2.2.4.1 Support check of diagnostic service identifier and adaptation of the Diagnostic message

This task shall remove the service identifier from the Diagnostic Message and check if the requested diagnostic service identifier is supported.

Dcm084: (Option) If an ECU receives a diagnostic request that contains a service ID which is in the range from 0x40 to 0x7F or in the range from 0xC0 to 0xFF the ECU shall not respond to such a request.

These range correspond to diagnostic response identifier.

The DSD will be triggered by the DSL if a new Diagnostic Message is recognized (by executing the Data Indication functionality).

The (incoming) Diagnostic Message shall be analyzed for the diagnostic service identifier (based on first byte of the Diagnostic Message) and the check of the supported services will be performed with the newly received diagnostic service identifier.

In this check the newly received diagnostic service identifier shall be searched in the "Service Identifier Table".

For performance reasons it might be necessary that the support check is done with a "Lookup table" functionality. In this "Service Identifier Table" all supported Service Ids of the ECU are predefined. The "Service Identifier Table" and the information about the supported services will be generated out of the configuration. More than one Service Identifier Table can be configured for selection. At one time only one Service Identifier Table can be active. Therefore the DSL functionality will provide the current Service Identifier Table (see chapter 8.7.3.1 DSL (Diagnostic Session Layer) Functions).

For the check the "Service Identifier Table" will be scanned for the newly received diagnostic service identifier. The diagnostic service identifier is not supported when it is not included in the "Service Identifier Table".

If the newly received diagnostic service identifier is supported, the associated interface will be called.

If the newly received diagnostic service identifier is not supported, the negative Response "Service not supported" (NRC 0x11) will be transmitted to the DSL.

The newly received diagnostic service identifier has to be stored for later usage of the DSD.

For example:

WriteDataByIdentifier (for writing VIN number):

1. A new Diagnostic Message will be received by the DSL.
(Diagnostic Message WriteDataByIdentifier = 0x2E,0xF1,0x90,0x57,0x30,0x4C,0x30,0x30,0x30,0x30,0x34,0x33,0x4D,0x42,0x35,0x34,0x31,0x33,0x32,0x36)

2. The DSL indicates a new Diagnostic Message with the "Data Indication" functionality to the DSD. In the Diagnostic message buffer the Diagnostic Message is stored (buffer = 0x2E,0xF1,0x90,..)

3. The DSD will execute a check of the supported services with the determined service identifier (first byte of buffer 0x2E) on the incoming Diagnostic message.

4. The incoming Diagnostic message will be stored into the DCM Variable Dcm_MsgContextType

Negative responses of following types will be handled:

- Service not supported (NRC 0x11)

The check of "not supported sub-functions" shall be done in the DSP.

7.2.2.4.2 Handling of „suppressPosRspMsgIndicationBit“

Dcm017: The suppressPosRspMsgIndicationBit is part of the Sub-function parameter structure (Bit 7 based on second byte of the Diagnostic Message).

[see ISO 14229 chapter 6.5: Server response implementation rules]

If the “suppressPosRspMsgIndicationBit” is TRUE the ECU shall NOT send a positive response message.

The “suppressPosRspMsgIndicationBit” shall be removed (by masking the Bit) from the Diagnostic message.

The information whether a suppression of a positive response is active shall be transported (between the layers) via the parameter Dcm_MsgContextType (Element: Dcm_MsgAddInfo).

In case of responsePending: suppressPosRspMsgIndicationBit shall be cleared. Since final response (negative /positive) is required then.

For the reason that the „suppressPosRspMsgIndicationBit” is only available if a service has a sub function, the „suppressPosRspMsgIndicationBit” handling shall only be done when the „ ***DcmDsdSidTabSubFuncAvail***” configuration is set for the newly received service identifier in the “Service Identifier Table”.

7.2.2.4.3 Verification functionality

The following verifications shall be executed:

- Verification of the Diagnostic Session (e.g. ‘Default session’, Extended session)
- Verification of the Service Security Access levels
- Verification of application environment / permission

Please note, that service shall only be accepted if ALL 3 verifications pass.

Verification of the Diagnostic Session:

The DiagnosticSessionControl service is used to enable different diagnostic sessions in the ECU (e.g. ‘Default session’, Extended session).

A diagnostic session enables a specific set of diagnostic services and/or functionality in the ECU. It furthermore enables a protocol depending data set of timing parameters applicable for the started session. This handling shall be done with the DiagnosticSessionControl (0x10) service. But the DSD shall perform a verification whether the execution of the requested service, NOT the DiagnosticSessionControl (0x10) service itself, is allowed in the current Diagnostic Session or not.

Therefore the current Diagnostic Session shall be asked for at the DSL by using the DSL function Dcm_GetSesCtrlType (). For more information please refer to chapter 8.4.2.2 Dcm_GetSesCtrlType.

The handling of the Diagnostic Session Control is not part of the DSD.

The Verification of the Diagnostic Session shall be performed with the newly received diagnostic service identifier.

Therefore for the “**DcmDsdSidTabSessionLevel**” configuration shall be searched in the “Service Identifier Table” (the newly received diagnostic service identifier).

When the newly received diagnostic service is allowed within the current Diagnostic Session the “Distribution of Diagnostic Message to data processor” can be started. If the newly received diagnostic service is not allowed with the current Diagnostic Session, the negative Response “Service not supported in active session” (NRC 0x7F) will be transmitted to the DSL.

Verification of the Service Security Access levels:

The purpose of the Security Access level handling is to provide a possibility to access data and/or diagnostic services, which have restricted access for security, emissions, or safety reasons. This handling shall be done with the SecurityAccess (0x 27) service. The DSD shall perform a verification whether the execution of the requested service (NOT the SecurityAccess (0x27) service) is allowed in the current Security level or not.

Therefore, the DSL shall be asked for the current security level by using the DSL function Dcm_GetSecurityLevel().

The management security level is not part of the DSD.

Note: For some use cases (e.g. Service ReadDataByIdentifier where some DataIdentifier can be secure) it will be necessary that the application has to call the function Dcm_GetSecurityLevel().

The Verification of the Security Access levels shall be performed with the newly received diagnostic service identifier.

Therefore for the “**DcmDsdSidTabSecLevel**” configuration shall be searched in the “Service Identifier Table” (for the newly received diagnostic service identifier).

When the newly received diagnostic service is allowed with the current Security level the “Distribution of Diagnostic Message to data processor” can be started.

If the newly received diagnostic service is not allowed with the current Security level, the negative Response “Security access denied” will be transmitted to the DSL.

Negative responses of following types will be handled:

- Security access denied (NRC 0x33)
- Service not supported in active session (NRC 0x7F)

Verification of application environment / permission

The purpose of this functionality is, that right before processing the diagnostic message, the application is requested to check permission / environment. E.g in after-run ECU state, it might be not allowed to process OBD requests.

If configured (configuration parameter ***DcmRequestIndicationEnabled*** set to TRUE, see chapter 10.2 DCM global Configurations), DSD calls the function `Dcm_Indication()` and provides information about SID and other contents of diagnostic request to the application. If application allows, processing of diagnostic message is continued.

Otherwise (depended on application return):

- no response
- negative response with ConditionsNotCorrect (NRC 0x22))

7.2.2.4.4 Distribution of Diagnostic Message to data processor (DSP or AUTOSAR Software Component)

Task of this functionality is to search for the executable functionality of the DSP or the Application (basic SW modules via the standardized RTE Interface) for newly received diagnostic service identifier, to call the corresponding Application function and provide the Diagnostic Message/Data to this Application function.

Search for the executable functionality in the “Service Identifier Table”:

The “Service Identifier Table” includes an entry for the ***DcmDsdSidFunction*** element which determines the function pointer to the function of the DSP.

This entry is configurable. Names for the ***DcmDsdSidFunction*** are listed in Table 5: List of function naming examples for requests of ISO14229-1 and Table 6: List of function naming examples for requests of ISO15031-5.

7.2.2.4.4.1 Assemble positive or negative response

When the Application Software (or the DSP) has finished the execution of the requested Diagnostic Service the response has to be assembled.

The execution of the application can have the results:

- positive Result
- negative Result

Following possible Responses can be assembled:

- positive Response
- negative Response
- no Response (in the case of Suppression of responses)

Positive Response:

The positive Result of the DSP functionality is indicated by the function `Dcm_ProcessingDone(Dcm_MsgContextType)`.

The parameter “`Dcm_MsgContextType`” comprises the diagnostic (response) message.

In the parameter “`Dcm_MsgContextType`” the response service identifier and the response data stream (returned by the Application) will be added.

Therefore the `Dcm_MsgContextType` will be transferred into a (response) buffer and the service identifier will be added at the first byte of the buffer.

In the next execution step the “Initiate transmission” functionality will be executed.

Negative Response:

The negative Result (error case e.g. conditions not correct) of the DSD functionality is indicated by the function `Dcm_SetNegResponse(Dcm_MsgContextType , NRC)` followed by the `Dcm_ProcessingDone(Dcm_MsgContextType)` function.

The parameter “`Dcm_MsgContextType`” comprises the diagnostic (request) message.

The parameter “NRC” represents the Negative Response Code.

For the allowed NRC of the executed Service ID please refer to the specification of the service in ISO14229-1 (UDS) (see chapter 4.2.4 Response code parameter definition Table 12) and ISO15031-5 (OBD/CARB). The application has to take care of the correct usage of NRC of the executed Service ID.

The valid set of the Negative Response Codes for the Application is defined in the configuration (based on the table `Dcm_NegativeResponseCode`).

Furthermore the service identifier of the request will be added to the Negative Response Code (see “Distribution of Diagnostic Message to data processor”) and the “Initiate transmission” functionality will be executed.

Negative responses of following types will be handled:

All Negative Response Codes supported from the Application and defined in the table `Dcm_NegativeResponseCode`.

Suppression of Responses:

The Suppression of Responses can be active in the following cases:

- Positive Result of execution and indication of “`suppressPosRspMsgIndicationBit`” in Request
- Negative Result of the execution while Functional Addressing is active and one of the Negative Response Codes 0x11 or 0x12 or 0x31 occurs

Indication of “`suppressPosRspMsgIndicationBit`” in Request:

In the case that the `suppressPosRspMsgIndicationBit` was indicated in the functionality “Handling of `suppressPosRspMsgIndicationBit`” (stored in the Variable `Dcm_MsgContextType` (Element: `Dcm_MsgAddInfo`)), the suppression of Positive Responses is active.

Functional Addressing

Dcm001: In the case of a Negative Result of execution and Functional Addressing is active the suppression of following Negative Responses is active:

0x11 (DCM_E_SERVICENOTSUPPORTED) indicated by DSD functionality
"Removal and Support check of diagnostic service identifier"

0x12 (DCM_E_SUBFUNCTIONNOTSUPPORTED)

0x31 (DCM_E_REQUESTOUTOFRANGE)

The information if Functional Addressing is active is available in the MsgContextType.

7.2.2.4.5 Initiate transmission

Purpose of the Initiate transmission task is to forward the diagnostic (response) message (positive or negative response) to the DSL which will bring it further to the PDU Router.

This will be done by executing a DSL transmit functionality.

The DSL will send the data to the PDU Router and will receive a confirmation. This confirmation will be forwarded to the DSD. The DSD will forward the Confirmation via the function Dsp_DcmConfirmation (Dcm_MsgContextType) to the DSP.

In the case that no diagnostic (response) message shall be send (Suppression of Responses) the DSL Functionality "No Transmit" shall be executed.

In this case no Data Confirmation is send from the DSL towards the DSD. But the DSP is waiting on the execution of the Dsp_DcmConfirmation (Dcm_MsgContextType) functionality. So the DSD has to send this indication to the DSP.

The processing of one Diagnostic Message of the Diagnostic Service Dispatcher (DSD) is finished with the "Dsp_DcmConfirmation" action.

This means that in any following cases:

- Positive Response
- Negative Response
- Suppressed Positive Response
- Suppressed Negative Response

the DSD will finish with sending the "Dsp_DcmConfirmation" to the Application.

7.2.3 DSP (Diagnostic Service Processing)

7.2.3.1 Introduction

The DSP is mainly a container for completely implemented diagnostic services that are common amongst the different applications (e.g. access to the fault data) and thus do not need to be implemented by the application.

The DSP provides an interface to the DSD and implements following diagnostic services:

Emissions-related diagnostic services:

- legislated emission related diagnostic service for reading Current Diagnostic Data data (service 0x01) (Emissions-related applications only)
- legislated emission related diagnostic service for reading freeze-frame data (service 0x02) (Emissions-related applications only)
- legislated emission related diagnostic services for accessing the fault data (service 0x03 and 0x07) (Emissions-related applications only)
- legislated emission related diagnostic service for clearing fault data (service 0x04) (Emissions-related applications only)

Unified Diagnostic Services:

Diagnostic event manager interface

:

- Clear Diagnostic Information (service 0x14)
- Read DTC Information (service 0x19)
- Control DTC Setting (service 0x85)

Diagnostic communication and security:

- Diagnostic Session Control (service 0x10)
- Tester Present (service 0x3E)
- Security Access (service 0x27)

Diagnostic data access:

- ReadDataByIdentifier (service 0x22)
- ReadDataByPeriodicIdentifier (service 0x2A)
- WriteDataByIdentifier (service 0x2E)
- InputOutputControlByIdentifier (service 0x2F)
- RoutineControl (service 0x31)
- ReadScalingDataByIdentifier (service 0x24)
- DynamicallyDefineDataIdentifier (service 0x2C)

Reset control:

- EcuReset (service 0x11)

Later releases of this specification provide the option to extend the number of services implemented completely.

With provisions of an extended number of diagnostic services already implemented within DSP the effort for each individual SW – Component implementation is considerably reduced as well as a higher level of standard compliance is achieved.

7.2.3.2 Use cases

The DSP implements a set of diagnostic services required to provide basic diagnostic functionality. This implementation covers following diagnostic functionality:

- diagnostic session transitions
- locking and unlocking of specific diagnostic service per Security Access
- reading of ECU resident DTCs and associated data
- deletion of ECU resident DTCs and associated data
- enabling and disabling of DTC storage
- reading and writing of diagnostic data
- Control of diagnostic routine

7.2.3.3 Interaction with other modules

The DSP interacts with the following modules:

- **Dcm029:**
DEM: for implementation of the services to read and clear fault data and freeze-frame data.
- SW-Components:
for implementation of the security access algorithms when the specific security encryption method is not known by the DSP and therefore provided by the application
for access to diagnostic data and diagnostic routines
- Internally (within DCM): functional block DSP interacts with functional block DSL

7.2.3.4 Sub-function and format check

Dcm026:

Before the DSP performs the requested command, following checks shall be executed:

- Check whether a specific sub-function is supported as described in [12] and the service specific configuration (for details pls. refer to 10.5 DSP (Diagnostic Service Processor).
If the requested sub-function is not configured `Dcm_SetNegResponse()` shall be called to set the negative response code: "SubFunctionNotSupported" (NRC 0x12).
- Check for appropriate message length and structure:
If the message length and the structure of the received diagnostic message does NOT comply to [12], `Dcm_SetNegResponse()` shall be called to set the negative response code: "IncorrectMessageLengthOrInvalidFormat" (NRC 0x13).

Note: It is up to the implementation in which detail the format check might be executed and depends on the level of detail the diagnostic data description provides at compile time.

7.2.3.5 Error manager interface

ISO 14229-1 as well as ISO 15031-5 provide a variety of methods to gather data from the error memory of an ECU. This error manager interface implements the diagnostic services defined by the above mentioned international standards. The API function calls used for assembling the respective diagnostic responses are provided by the Diagnostic Event Manager (DEM), which is a Basic Software module included in the AUTOSAR Layered Architecture.

DTC Origin:

When using interfaces with DEM, the following DTC origin have to be used :

Dcm077: For emission related services, all services will access always the DEM_DTC_ORIGIN_PRIMARY_MEMORY, except service \$0A which shall access a separate memory section DEM_DTC_ORIGIN_PERMANENT_MEMORY

Dcm078: For UDS, the DTC origin in the following table shall be used:

Service	Sub-service	DTC Origin
\$14		DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	01	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	02	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	03	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	04	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	05	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	06	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	07	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	08	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	09	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	0A	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	0B	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	0C	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	0D	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	0E	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	0F	DEM_DTC_ORIGIN_MIRROR_MEMORY
\$19	10	DEM_DTC_ORIGIN_MIRROR_MEMORY
\$19	11	DEM_DTC_ORIGIN_MIRROR_MEMORY
\$19	12	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	13	DEM_DTC_ORIGIN_PRIMARY_MEMORY
\$19	14	DEM_DTC_ORIGIN_PRIMARY_MEMORY

\$19	15	DEM_DTC_ORIGIN_PERMANENT_MEMORY
\$31	01 FF 02	DEM_DTC_ORIGIN_MIRROR_MEMORY

7.2.3.5.1 Usage of Paged Buffer mechanism

Dcm038: If Paged Buffer mechanism is used the DSP shall determine the overall response length before any data is passed to DSD or DSL respectively. This requirement needs to be implemented as segmented diagnostic data transmission on CAN using ISO 15765-2 transport layer requires to provide the overall length of the complete data stream in the very first CAN frame of the respective data transmission.

(pls. refer to section 7.2.1.4.2.7 for details about Paged Buffer mechanism).

7.2.3.5.2 Basic sequence

When receiving a function call from DSD requiring DSP to process a diagnostic service request, it shall always carry out following basic process steps:

- analyze of the received request message
- check format and whether the addressed sub-function is supported (see section 7.2.3.4)
- either acquire data from DEM or execute a specific DEM function call

Dcm039:

- assemble the response message excluding response service identifier and determine the response message length (if PagedBuffer mechanism is used, overall message length has to be determined first, before passing the first response message data packet to DSD / DSL)
- confirm completion of request processing with function call Dcm_ProcessingDone()

Following sections describe either how the respective data shall be retrieved from DEM or how a specific DEM function is executed when receiving a specific diagnostic request message.

If an error occurs when processing a diagnostic request within DSP, the function Dcm_SetNegResponse() shall be executed before Dcm_ProcessingDone() is called. Following Negative Response Codes may be set when one of the following failures occurs:

- API call to the DEM causes a return value <> OK : NRC 0x22
- Analysis of the request message results in:
 - formatting or length failure: NRC 0x13
 - sub-function not supported: NRC 0x12
 - sub-function not supported in active diagnostic session: NRC: 0x7F
 - other unsupported message parameters : NRC: 0x31

7.2.3.5.3 Emission related diagnostic services according ISO15031-5

A.) Request powertrain Freeze Frame data - Request DTC of Freeze Frame

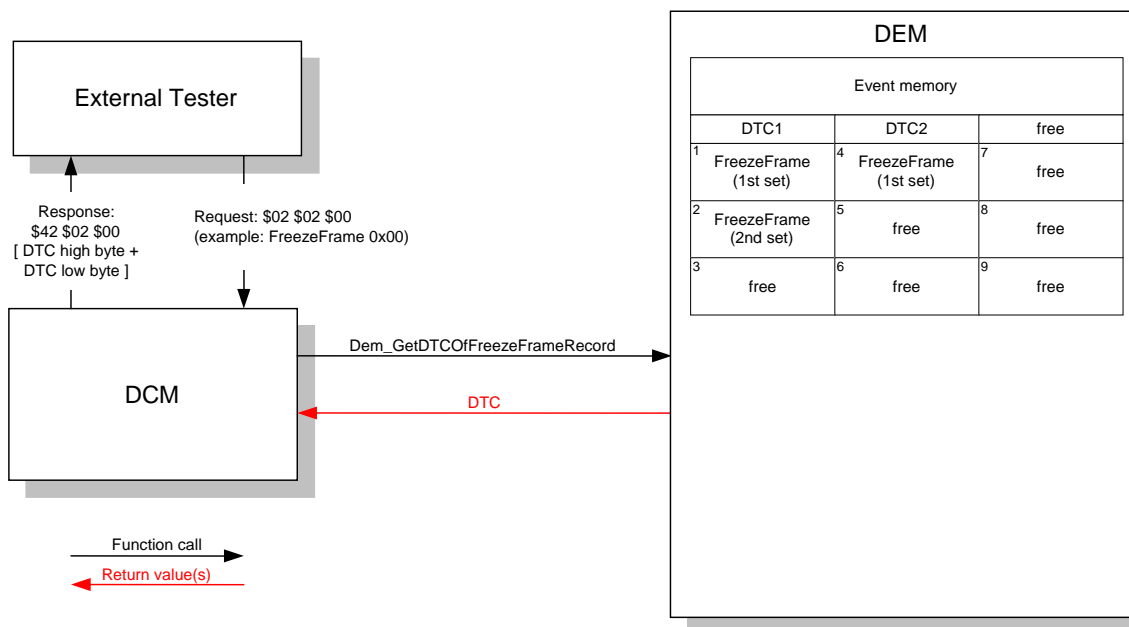


Figure 8: Request powertrain freeze frame data – Request DTC of Freeze Frame

- A.1. An external tester requests a DTC for a specific FreezeFrame by sending [0x02 0x02 0x00].
- A.2. The DSP calls `Dem_GetDTCOfFreezeFrameRecord`.
- A.3. The DEM returns the corresponding DTC to the DSP afterwards.
- A.4. The DSP generates a response message including PID (0x02), Freeze Frame number and the DTC (high byte and low byte) that caused the respective Freeze Frame to be stored.
- A.5. The DCM sends the response message back to the external tester.

Note: The external test tool may request maximally for 3 Freeze Frames within one request. That means steps A.2. and A.3. described above may be successively executed several times.

B.) Request powertrain freeze frame data - Request supported PIDs of a particular Freeze Frame

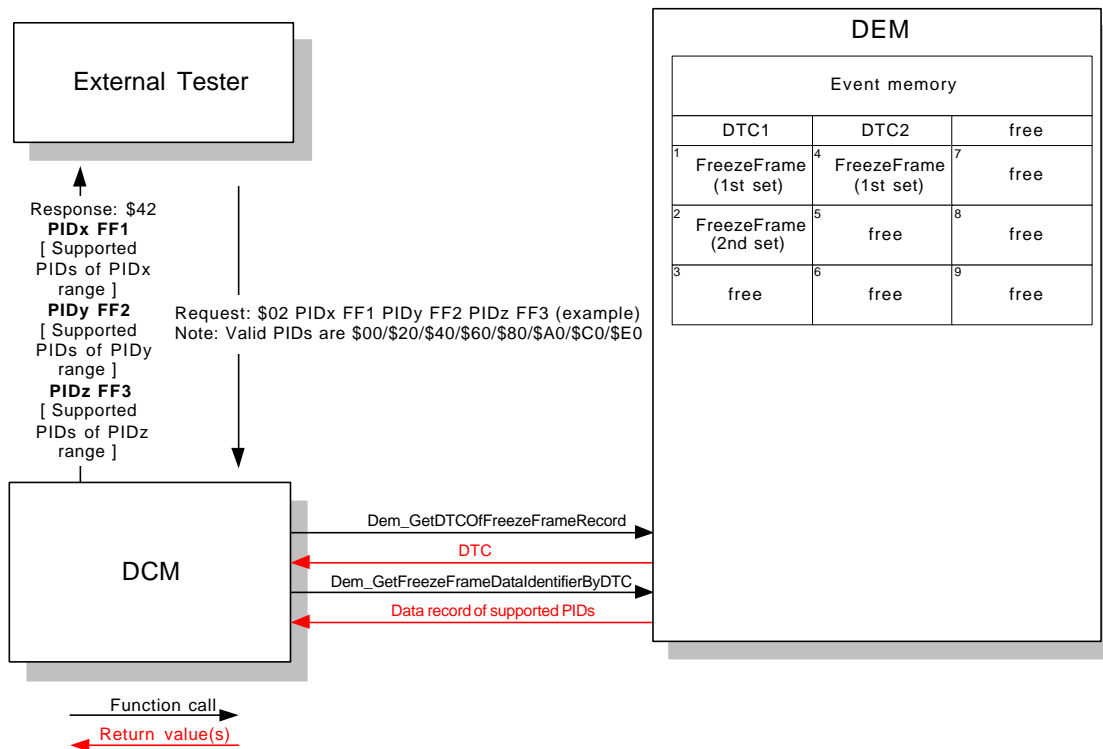


Figure 9: Request powertrain freeze frame data – Request supported PIDs of a particular Freeze Frame

- B.1. An external tester requests the supported PIDs for a specific FreezeFrame by sending 0x02 and the PIDs, which request the PIDs supported by a specific Freeze Frame (maximum: 3 PIDs per request).
- B.2. The DSP calls Dem_GetDTCOfFreezeFrameRecord. It is theoretically possible to request supported PIDs from different Freeze Frames in one request. In that case the DSP has to request the DTC for each of the requested FreezeFrames via Dem_GetDTCOfFreezeFrameRecord.
- B.3. The returned DTC is used to call Dem_GetFreezeFrameDataIdentifierByDTC.
- B.4. The returned information is interpreted by the DSP and a 32bit bit-mask with initially all bits set to zero (0) is generated based on the following scheme:
 - If PID 0x00 is requested set each bit in the mask to one (1) for each PID in the list of supported PIDs starting from offset 0xF501 through 0xF51F. If the list of supported PIDs contains more PIDs in the range from 0xF520...0xF5FF the last bit in the 32bit mask shall also be set to one (1).
 - If PID 0x20 is requested set each bit in the mask to one (1) for each PID in the list of supported PIDs starting from offset 0xF521 through 0xF53F. If the list of supported PIDs contains more PIDs in the range from 0xF540...0xF5FF the last bit in the 32bit mask shall also be set to one (1).
 - If PID 0x40 is requested set each bit in the mask to one (1) for each PID in the list of supported PIDs starting from offset 0xF541 through 0xF55F. If the list of supported PIDs contains more PIDs in the range from 0xF560...0xF5FF the last bit in the 32bit mask shall also be set to one (1).

Note: The method described above shall also be applied if following PIDs are requested: 0x60, 0x80, 0xA0, 0xC0, 0xE0. Each of these PIDs is associated with a PID range from 0x00 to 0x1F and the respective request PID is considered as the particular offset. For PID 0xE0 the exception applies that the last bit is to be set to zero (0) always, as the reserved OBD PID range can not be exceeded.

B.5. Finally the response message is generated which includes the requested bit-masks, representing the supported PIDs. After that the response message is sent back to the external tester by the DSP / DCM.

C.) Request powertrain freeze frame data - Request the data records assigned to a specific PID included in a specific Freeze Frame

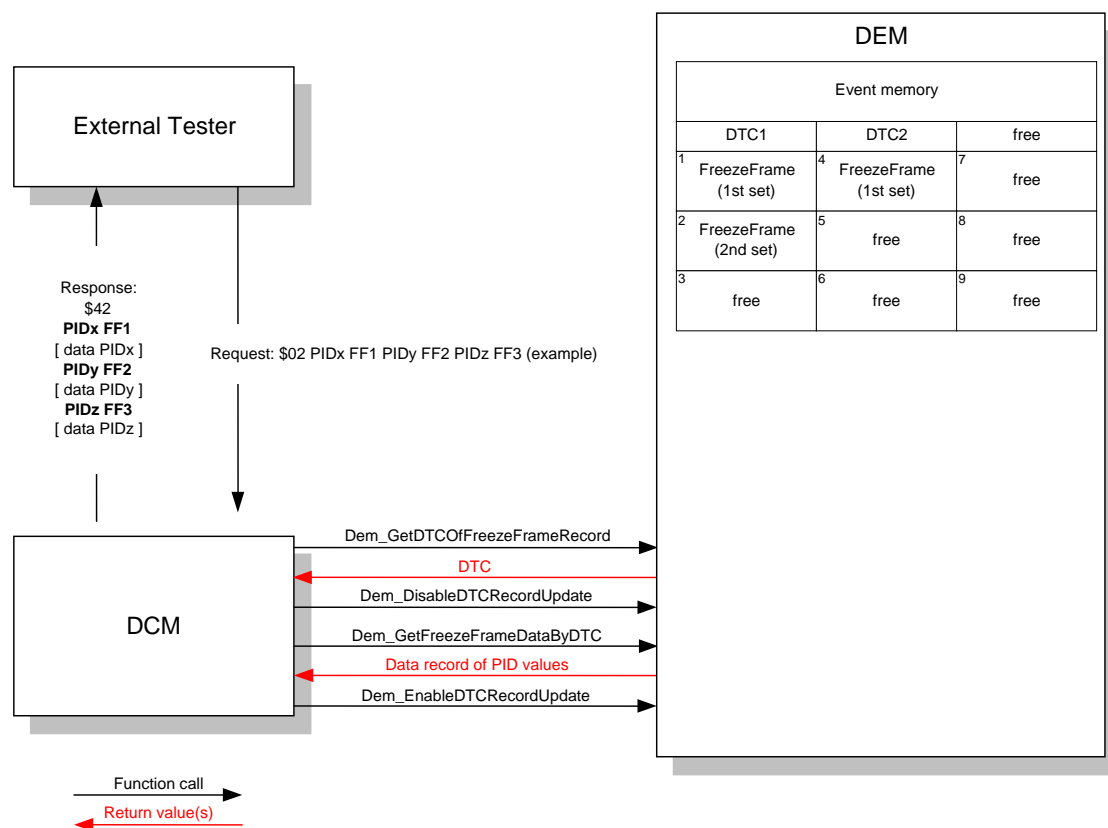


Figure 10: Request powertrain freeze frame data – Request data assigned to the PIDs of a particular Freeze Frame

- C.1. An external tester requests PIDs for specific Freeze Frames by sending 0x02 and the PIDs for the requested Freeze Frames (maximum: 3 PIDs per request).
- C.2. After that the DSP calls Dem_GetDTCOfFreezeFrameRecord. It is theoretically possible to request PIDs of different Freeze Frames in one request. In that case the DSP has to request the DTC for each of the requested FreezeFrames via Dem_GetDTCOfFreezeFrameRecord.

- C.3. The DSP has to call Dem_DisableDTCRecordUpdate (which is an exclusive area function concerning to BSW00434) to avoid the update of data values during reading them out.
- C.4. Further, the DSP requests the Freeze Frame data for the requested PIDs by calling Dem_GetFreezeFrameDataByDTC (loop for all PIDs).
- C.5. The response message generated by the DSP includes the respective PID, Freeze Frame Number and the associated data record for the requested FreezeFrame number.
- C.6. The response message is sent back to the external tester by the DSP / DCM.
- C.7. The DSP has to enable the update of data by calling Dem_EnabledDTCRecordUpdate.

D.) Request emission-related diagnostic trouble codes

- D.1. An external test tool requests an emissions-related ECU to report all stored emissions-related DTCs by sending the request 0x03.
- D.2. After that the DSP calls the function Dem_SetDTCFilter to retrieve the emissions-related DTCs with a “confirmed” status from the DEM error memory (upon receiving this function call the DEM compiles a list of DTCs that match the filter criteria)
- D.3. The DSP sequentially calls the Dem_GetNextFilteredDTC to retrieve the first / next DTC matching the criteria, in parallel the response is assembled.
- D.4. This function call is repeated as long as the function returns that no DTC is available that matches the filter criteria (return value equal to “NoMatchingDTC”).

E.) Clear/reset emission-related diagnostic information

Dcm004: An external test tool requests an emissions-related ECU to clear the error memory by sending the request 0x04. Upon receiving such a request DSP executes DEM function Dem_ClearDTC. The parameter of this DEM function “DTCGroup” shall indicate that all emissions-related DTCs are in scope. Therefore the function Dem_ClearDTC shall erase the emissions related DTCs and all related information from the fault memory

F.) Request emission-related diagnostic trouble codes detected during current or last completed driving cycle (diagnostic service 0x07)

- F.1. An external test tool requests an emissions-related ECU to report all pending emissions-related DTCs by sending the request 0x07.
- F.2. The DSP calls the function Dem_SetDTCFilter to retrieve the emissions-related DTCs with a “pending” status from the DEM error memory (upon receiving this function call the DEM compiles a list of DTCs that match the filter criteria)
- F.3. The DSP sequentially calls the Dem_GetNextFilteredDTC to retrieve the first / next DTC matching the criteria, in parallel the response is assembled.

F.4. This function call is repeated as long as the function returns that no DTC is available that matches the filter criteria (return value equal to "NoMatchingDTC")

7.2.3.5.4 Enhanced diagnostic services according ISO14229-1 (UDS)

Note: The total response length of the UDS response can be calculated by calling the following DEM functions (e.g. by calling in a loop) for the following subfunctions of service 0x19. (list not complete)

Subfunction of Service 0x19	DEM function for calculating the total response length of UDS response
0x02 (ReportDTCByStatusMask)	Dem_GetNumberOfFilteredDTC
0x03 (ReportDTCSnapshotIdentification)	Dem_SetDTCFilterForRecords
0x04 (ReportDTCSnapshotRecordByDTCNumber)	Dem_GetSizeOfFreezeFrame
0x05 (ReportDTCSnapshotRecordByRecordNumber)	Dem_GetSizeOfFreezeFrame
0x06 (ReportDTCExtendedDataRecordByDTCNumber)	Dem_GetSizeOfExtendedDataRecordByDTC
0x08 (ReportDTCBySeverityMaskRecord)	Dem_GetNumberOfFilteredDTC
0x0A (ReportSupportedDTCs)	Dem_GetNumberOfFilteredDTC
0x0F (ReportMirrorMemoryDTCByStatusMask)	Dem_GetNumberOfFilteredDTC
0x10 (ReportMirrorMemoryDTCExtendedDataRecordByDTCNumber)	Dem_GetSizeOfExtendedDataRecordByDTC
0x13 (ReportEmissionRelatedOBDDTCByStatusMask)	Dem_GetNumberOfFilteredDTC
0x14 (ReportDTCFaultDetectionCounter)	Dem_GetNumberOfFilteredDTC
0x15 (ReportDTCWithPermanentStatus)	not supported by DEM / DCM

A.) Clear Diagnostic Information (diagnostic service 0x14)

Dcm005: An external test tool requests an ECU to clear the error memory by sending the request 0x14 [DTC Group]. When the DSP receives such a request it executes DEM function Dem_ClearDTC with parameter DTCGroup equal to the value received in the request to remove either one DTC or a group of DTCs and all related information from the fault memory.

B.) Read DTC Information

Report number of DTCs matching a specific status mask or severity mask record, memory location and DTC type

Dcm007:

- B.1. An external test tool requests an ECU to report the number of DTCs matching a tester defined criteria by sending a request (SID 0x19) including one of the following sub-functions: 0x01, 0x07, 0x11 or 0x12.
- B.2. The very first step of the DSP is to determine the Status Availability Mask by calling DEM sub-function Dem_GetDTCStatusAvailabilityMask. This mask reflects the status bit supported by the respective ECU.
- B.3. After that the DSP calls Dem_SetDTCFilter to count the DTCs matching the requested status or severity mask record, memory location and DTC type.
- B.4. Then the DSP calls Dem_GetNumberOfFilteredDTC to retrieve the count of filtered DTCs.

C.) Read DTC Information

Report DTCs with the associated status information matching a specific status mask, memory location and DTC type

Dcm008:

An external test tool requests an ECU to report DTCs and the associated status, which match a tester defined status mask, DTC group and memory location, by sending a request (SID 0x19) including one of the following sub-functions: 0x02, 0x0A, 0x0F, 0x13

- C.1. The very first step of the DSP is to determine the Status Availability Mask by calling DEM sub-function Dem_GetDTCStatusAvailabilityMask. This mask reflects the status bit supported by the respective ECU.
- C.2. The DSP checks the value of DTCStatusMask. If equal to 0, the DSP answers positively with 0 DTC and the further steps (C.3, C.4, C.5 and C.6) are not processed. If different from 0, DSP process next steps :
- C.3. After that the DSP calls Dem_SetDTCFilter to filter on the DTCs matching the requested status, memory location and DTC type.
- C.4. Then the DSP calls Dem_GetNumberOfFilteredDTC to retrieve the count of filtered DTCs.
- C.5. The DSP sequentially calls the Dem_GetNextFilteredDTC to retrieve the first / next DTC and the associated DTC status
- C.6. Step C.5 shall be repeated until the function call returns a value indicating that no DTC is matching the filter criteria (return value equal to "NoMatchingDTC")

D.) Read DTC Information

DTCs with the associated status information matching a specific severity mask record

Dcm009:

- D.1. An external test tool requests an ECU to report DTCs and the associated status, which match a tester defined severity mask record, by sending a request (SID 0x19) including sub-function 0x08
- D.2. The very first step of the DSP is to determine the Status Availability Mask by calling DEM sub-function Dem_GetDTCStatusAvailabilityMask. This mask reflects the status bit supported by the respective ECU.
- D.3. After that the DSP calls Dem_SetDTCFilter to filter on the DTCs matching the requested severity mask record.
- D.4. Then the DSP calls Dem_GetNumberOfFilteredDTC to retrieve the count of filtered DTCs.
- D.5. The DSP sequentially calls the Dem_GetNextFilteredDTC to retrieve the first / next DTC and the associated DTC status.
- D.6. The DSP calls Dem_GetSeverityOfDTC to get the severity information of a DTC retrieved in D.5.
- D.7. The DSP calls Dem_GetViewIDofDTC to get the viewID of a DTC retrieved in D.5.
- D.8. Step D.5 to D.7 are to repeat until the function call returns that no DTC is available matching the filter criteria (return value equal to "NoMatchingDTC")

E.) Read DTC Information
Report all Extended Data Records for a particular DTC

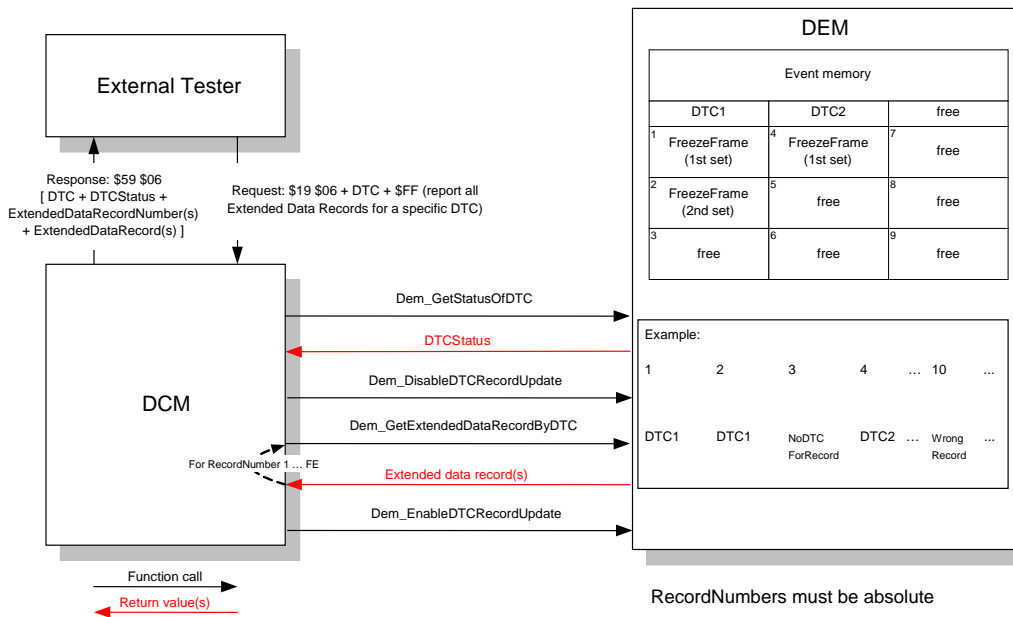


Figure 11: Request all Extended Data Records for a particular DTC

- E.1. By sending 0x19 0x06 + DTC + 0xFF an external tester requests the corresponding Extended Data Records for a specific DTC.
- E.2. The DSP calls Dem_GetStatusOfDTC to request status information for the requested DTC.
- E.3. The corresponding status is returned by the DEM.
- E.4. The DSP has to call Dem_DisableDTCRecordUpdate (which is an exclusive area function concerning to BSW00434) to avoid the update of data values during reading them out.
- E.5. To request the Extended Data Record for a particular DTC the DSP has to call Dem_GetExtendedDataRecordByDTC. This function has to be called in a loop until all Extended Data Record Numbers are processed.
- E.6. The corresponding Extended Data Record(s) are/is returned by the DEM.
- E.7. A response message with the DTC, the DTC status, the Extended Data Record Number(s) and the Extended Data Record(s) is generated by the DSP/DCM accordingly and sent back to the external tester.
- E.8. The DSP has to enable the update of data by calling Dem_EnabledDTCRecordUpdate.

F.) Read DTC Information
Report one specific Extended Data Records for a particular DTC

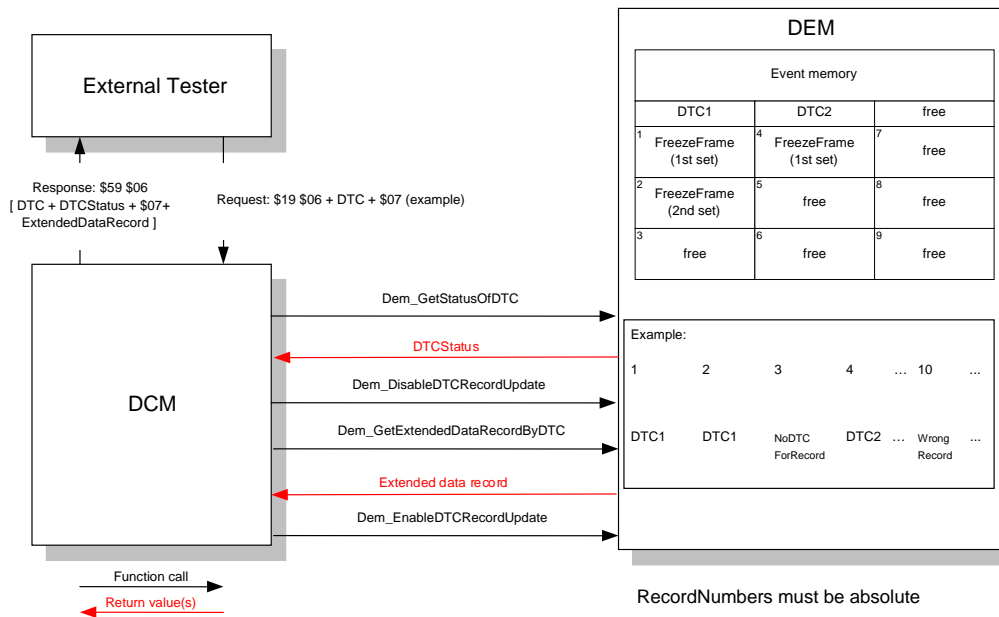


Figure 12: Request one specific Extended Data Records for a particular DTC

- F.1. By sending 0x19 0x06 + DTC + 0x07 (example) an external tester requests one particular Extended Data Records for a DTC.
- F.2. The DSP calls Dem_GetStatusOfDTC to request status information for the requested DTC.
- F.3. The corresponding status is returned by the DEM.
- F.4. The DSP has to call Dem_DisableDTCRecordUpdate (which is an exclusive area function concerning to BSW00434) to avoid the update of data values during reading them out.
- F.5. To request the Extended Data Record for a particular DTC a particular Extended Data Record Number the DSP has to call Dem_GetExtendedDataRecordByDTC.
- F.6. The corresponding Extended Data Record is returned by the DEM.
- F.7. A response message with the DTC, the DTC status, the Extended Data Record Number and the Extended Data Record is generated by the DSP / DCM accordingly and sent back to the external tester.
- F.8. The DSP has to enable the update of data by calling Dem_EnableDTCRecordUpdate.

G.) Read DTC Information
Report DTC Snapshot Record Identification

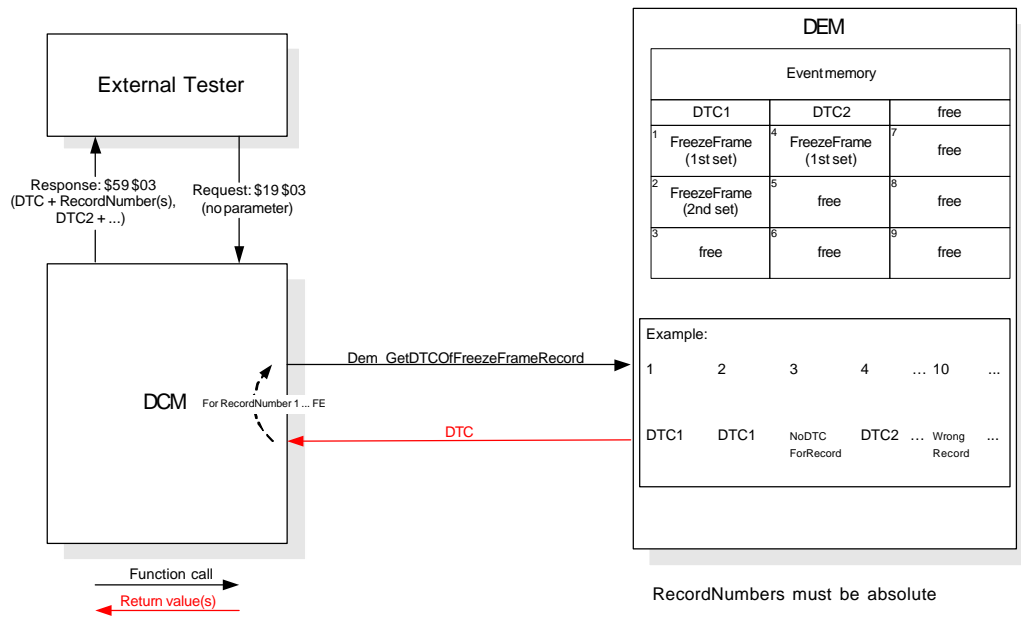


Figure 13: Request DTC Snapshot Record Identification

- G.1. By sending 0x19 0x03 an external tester requests the corresponding DTCs for all Freeze Frame records present in an ECU.
- G.2. The DSP calls Dem_GetDTCOfFreezeFrameRecord. With this call the DSP requests the corresponding DTC for a specific FreezeFrame record number. Dem_GetDTCOfFreezeFrameRecord has to be called by the DSP in a loop until all record numbers (No. 00 – 255) are processed. The DEM function Dem_SetDTCFilterForRecords shall be used to calculate the loop counter and the total response length of the UDS response.
- G.3. A response message with a list of DTCs and their associated record numbers is generated by the DSP / DCM accordingly and sent back to the external tester.

H.) Read DTC Information
Report DTC Snapshot Record By DTC Number

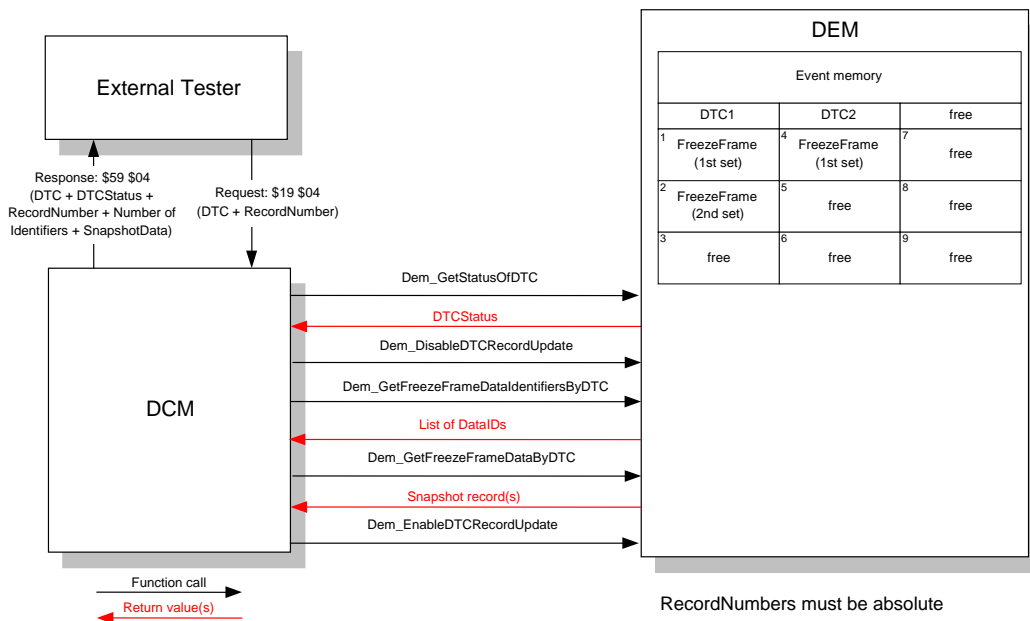


Figure 14: Request DTC Snapshot Record by DTC Number

- H.1. By sending 0x19 0x04 an external tester requests either one or all snapshot record(s) for a specific DTC.
- H.2. The DSP requests the status of a specific DTC by calling Dem_GetStatusOfDTC.
- H.3. The status of the requested DTC is returned to the DSP.
- H.4. The DSP has to call Dem_DisableDTCRecordUpdate (which is an exclusive area function concerning to BSW00434) to avoid the update of data values during reading them out.
- H.5. The DSP calls Dem_GetFreezeFrameDataIdentifierByDTC.
- H.6. The DEM returns the corresponding list of DataIDs included in the Freeze Frame the list was requested for.
- H.7. The DSP calls Dem_GetFreezeFrameDataByDTC to request the snapshot record assigned to one specific DataID of the list reported before. The DSP has to process the complete list of DataIDs (loop for all DataIDs).
- H.8. The DSP generates the response message and adds the DTC number including status, the number of DataIDs per Freeze Frame record, every single DataID and the associated data record(s).
- H.9. The response message is sent back to the external tester by the DCM.
- H.10. The DSP has to enable the update of data by calling Dem_EnableDTCRecordUpdate.

Note: The total response length of the UDS response can be calculated by calling the DEM function Dem_GetSizeOfFreezeFrame in a loop for all DataIDs.

I.) Read DTC Information

Report DTC Snapshot Record By Snapshot Record Number

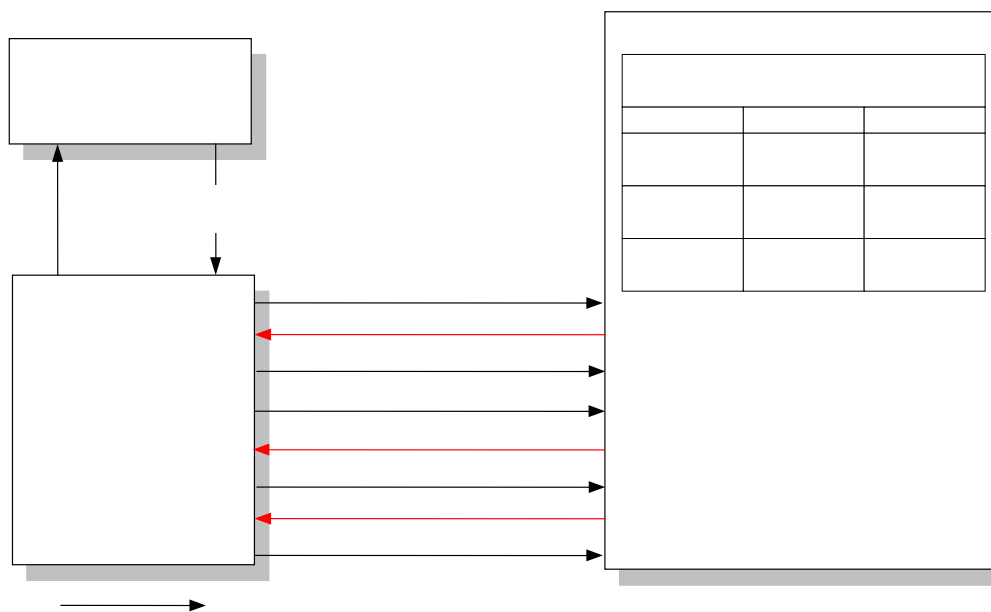


Figure 15: Request DTC Snapshot Record by Snapshot Record Number

- I.1. By sending 0x19 0x05 an external tester requests Freeze Frame information for a specific FreezeFrame record number.
- I.2. The DSP calls Dem_GetDTCOfFreezeFrameRecord.
- I.3. The DEM returns the corresponding DTC to the DSP.
- I.4. The DSP has to call Dem_DisabledDTCRecordUpdate (which is an exclusive area function concerning to BSW00434) to avoid the update of data values during reading them out.
- I.5. The DSP calls Dem_GetFreezeFrameDataIdentifierByDTC which returns a list of DataIDs associated to the requested DTC.
- I.6. The DSP calls Dem_GetFreezeFrameDataByDTC to request the corresponding data record(s) for the requested DataIDs (loop for all DataIDs).
- I.7. The DSP generates the response message and adds the DTC number including status, the number of DataIDs, every single DataID and the associated data record(s).
- I.8. The response message is sent back to the external tester by the DSP / DCM.
- I.9. The DSP has to enable the update of data by calling Dem_EnabledDTCRecordUpdate.

Note: The total response length of the UDS response can be calculated by calling the DEM function Dem_GetSizeOfFreezeFrame in a loop for all DataIDs.

J.) Read DTC Information

First Test Failed / First Confirmed DTC and associated status information

Most Recent Test Failed / Most Recent Confirmed DTC and associated status information

- J.1. An external test tool requests an ECU to report DTCs and the associated status, which match a tester defined occurrence criterion, by sending a request (SID 0x19) including one of the following sub-functions 0x0B, 0x0C, 0x0D, 0x0E.
- J.2. The very first step of the DSP is to determine the Status Availability Mask by calling DEM sub-function Dem_GetDTCStatusAvailabilityMask. This mask reflects the status bit supported by the respective ECU.
- J.3. After that the DSP calls the function Dem_GetDTCByOccurrenceTime to retrieve the DTC matching the requested criterion.
- J.4. The DSP calls the function Dem_GetStatusOfDTC to retrieve the current DTC status for the DTC returned by function Dem_GetDTCByOccurrenceTime.

K.) Control DTC Setting

K.1. An external test tool requests an ECU to either disable or enable DTC storage in the ECUs error memory by sending a request (SID 0x85) including one of the following sub-functions 0x01, 0x02.

K.2. Depending on the sub-function value received either Dem_DisableDTCStorage or Dem_EnabledDTCStorage is called.

7.2.3.6 Diagnostic communication and security

7.2.3.6.1 Diagnostic Session Control

Dcm023: The DiagnosticSessionControl service is used to enable different diagnostic sessions in the server. A diagnostic session enables a specific set of diagnostic services and/or functionality in the server.

Dcm_DcmDiagnosticSessionControl() function is called by DSD, when DiagnosticSessionControl service is recognized (SID set to 0x10), this service provided in DSP module and configured in DCM identifier table.

Inside this function DSP examines first, if request length is correct, otherwise error code is set to IncorrectMessageLengthOrInvalidFormat (NRC 0x13).

If request length fits, DSP checks if the requested subfunction value (diagnostic session type) is configured in the ECU (see configuration parameter **DcmDspSessionLevel**).

If requested subfunction value is not configured, DSP set error code to SubFunctionNot-Supported (NRC 0x12).

If requested subfunction value is configured, the following next steps are processed.

The next processing is done even if the requested session type is equal to the already running session type. In this way it is specified in the ISO ([11] chapter 8.2).

Dcm045:

Call the application callback function (see **Dcm_GetSesChgPermission()**) to examine environment settings and get the permission of application.

If application doesn't allow, error code is set to ConditionsNotCorrect (NRC 0x22).

If errors are detected, **Dcm_SetNegResponse()** is called, with the detected error code. Then **Dcm_ProcessingDone()** is called.

If no errors are detected **Dcm_ProcessingDone()** is called. The response is sent to the tester. The sent confirmation function will process following

- set the diagnostic session type (**Dcm_SetSesCtrlType()**)
- set the new timing parameters (P2ServerMax, P2ServerMax*) (see configuration parameters **DcmDspSessionP2ServerMax** and **DcmDspSessionP2StarServerMax**) (setting will be done through DCM internal API)

Referenced configuration parameters can be found in chapter 10.5 DSP (Diagnostic Service Processor). Referenced API is available in chapter 8 API specification. Sequence diagram is available in chapter 9.4.2.1 Process Diagnostic Session Control.

7.2.3.6.2 Tester Present

This service is used to keep one or multiple servers in a diagnostic session being different than the defaultSession.

Dcm_DcmTesterPresent() function is called by DSD, when TesterPresent service is recognized (SID set to 0x3E), this service provided in DSP module and configured in DCM identifier table.

Inside this function DSP examines first, if request length is correct, otherwise a service internal error code is set to IncorrectMessageLengthOrInvalidFormat (NRC 0x13).

Following subfunction values are supported: 0x00 and 0x80.

If received subfunction value is not equal to those values, DSP set error code to SubFunctionNot-Supported (NRC 0x12).

If errors are detected, **Dcm_SetNegResponse()** is called, with the detected error code. Then **Dcm_ProcessingDone()** is called.

Referenced API is available in chapter 8 API specification.

Sequence diagram is available in chapter 9.4.2.2 Process Tester Present.

7.2.3.6.3 SecurityAccess

Dcm021: The purpose of this service is to provide a means to access data and/or diagnostic services, which have restricted access for security, emissions, or safety reasons.

Dcm_DcmSecurityAccess() function is called by DSD, when SecurityAccess service is recognized (SID set to 0x27), this service is provided in DSP module and configured in DCM identifier table.

Inside this function DSP examines first, if request length is correct, otherwise a service internal error code is set to IncorrectMessageLengthOrInvalidFormat (NRC 0x13).

If the request length is correct, DSP checks if the requested subfunction value (access type) is configured in the ECU (see configuration parameter **DcmDspSessionLevel**).

If requested subfunction value is not configured, DSP set error code to SubFunctionNot-Supported (NRC 0x12).

If requested subfunction value is configured and a service with subfunction type request seed (= odd value) has been received, DSP examines, if the requested access type is already active (**Dcm_GetSecurityLevel()**).

If the access type is already active, DSP sets the seed content to 0x00 and finishes with **Dcm_ProcessingDone()**.

In the other case (access type is not active or “send key” request), the following application calls are processed:

- **Dcm_GetSeed()** (in case “request seed” is received)
- **Dcm_CompareKey()** (in case “send key” is received)

The following list gives as an example, which errors can be detected by the security access service and stored in the error code information:

- RequestSequenceError (NRC 0x24), when invalid access type is send at “send key”
- RequiredTimeDelayNotExpired (NRC 0x37), when time delay is active (see configuration parameter **DcmDspSecurityDelayTime**)
- ExceedNumberOfAttempts (NRC 0x36), when number of attempts to get security access exceeds (see configuration parameter **DcmDspSecurityNumAttLock**)
- InvalidKey (NRC 0x35), when invalid key is send at “send key”

If errors are detected, **Dcm_SetNegResponse()** is called, with the detected error code. Then **Dcm_ProcessingDone()** is called.

If no errors are detected, DSP sets the new access type (**Dcm_SetSecurityLevel()**) and finishes with **Dcm_ProcessingDone()**.

Referenced configuration parameters can be found in chapter 10.5 DSP (Diagnostic Service Processor).

Referenced API is available in chapter 8 API specification.

Sequence diagram is available in chapter 9.4.2.3 Process Security Access.

7.3 Error classification

This section describes how the DCM module has to treat the several error classes that may happen during the life cycle of this basic software.

Dcm047: The general requirements document of AUTOSAR [1] specifies that all basic software modules shall distinguish (according to the product life cycle) two error types:

- Development errors: those errors shall be detected and fixed during development phase. In most cases, those errors are software errors. The detection of errors that shall only occur during development can be switched off for production code (by static configuration namely preprocessor switches).
- Production errors: those errors are hardware errors and software exceptions that cannot be avoided and are expected to occur in production (i.e. series) code.

Dcm012: DCM provides no Production-Errors:

Diagnostic-Communication-Errors are handled directly in the ISO-Protocols by NRCs.

Dcm044: The used return values shall be the same for development and production. Only the values given by this specification shall be used.

Dcm040: The following errors and exceptions shall be detectable by the DCM depending on its build version (development/production mode). Development error values are of type uint8.

<i>Type or error</i>	<i>Relevance</i>	<i>Related error code</i>	<i>Value</i>
Application-Interface: Timeout	Development	DCM_E_INTERFACE_T IMEOUT	0x01
Application-Interface: Return-value out of range	Development	DCM_E_INTERFACE_V ALUE_OUT_OF_RANGE	0x02
Application-Interface: Buffer Overflow	Development	DCM_E_INTERFACE_B UFFER_OVERFLOW	0x03
Application-Interface: Protocol mismatch	Development	DCM_E_INTERFACE_P ROTOCOL_MISMATCH	0x04
Internal: DCM not initialized	Development	DCM_E_UNINIT	0x05
DCM API function with invalid input parameter	Development	DCM_E_PARAM	0x06

Dcm041: Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the DCM implementation specification. The classification and enumeration shall be compatible to the errors listed above

7.4 Error detection

Dcm042: The detection of development errors is configurable (*ON / OFF*) at pre-compile time.

The switch *DcmDevErrorDetect* (see chapter 10 Configuration specification) shall activate or deactivate the detection of all development errors.

Dcm043: If the development error detection is enabled for this module, for every request except *Dcm_Init*, it shall be ensured that the DCM is already initialized. Detected errors shall be reported to the Debug Error Tracer.

Dcm048: If the *DcmDevErrorDetect* switch is enabled API parameter checking is enabled except for *Dcm_Init*. The detailed description of the detected errors can be found in chapter 7.3 Error classification.

7.5 Error notification

Dcm049: Detected development errors will be reported to the error hook of the Development Error Tracer (DET) if the pre-processor switch *DcmDevErrorDetect* is set (see chapter 10).

Dcm050: The Development Error Tracer module is just help for BSW development and integration. It must not be contained inside the production code. The API is defined, but the functionality can be chosen and implemented according to the development needs (e.g. errors count, send error information via a serial interface to an external logger, and so on).

Dcm051: To report development errors the DCM shall use the Development Error Tracer service:

```
void Det_ReportError(uint16 ModuleId, uint8 InstanceId, uint8
                    ApiId, uint8 ErrorId)
```

Where:

- ModuleId: Module ID of calling module (DCM module ID = 0x35)
- InstanceId: The identifier of the index based instance of a module, starting from 0, If the module is a single instance module it shall pass 0 as the InstanceId. (DCM instance ID = 0x00)
- ApiId: ID of API service in which error is detected (defined in SWS of calling module)
- ErrorId: ID of detected development error (defined in SWS of calling module). It is specified in chapter 7.3 Error classification of this present document. However, its value can be #defined externally in the module's header file.

Dcm052: The header file of the DCM, DCM.h, shall provide a module ID called DCM_MODULE_ID sets to the value 0x35.

8 API specification

8.1 General

Dcm059: The same intention, logical contents or semantic shall be placed in one parameter only. (There must not be several parameters with the same intention, logical contents or semantic).

Dcm060: A parameters name must be unique per module. If the parameter is exported it must be unique to all modules using this parameter.

Dcm061:

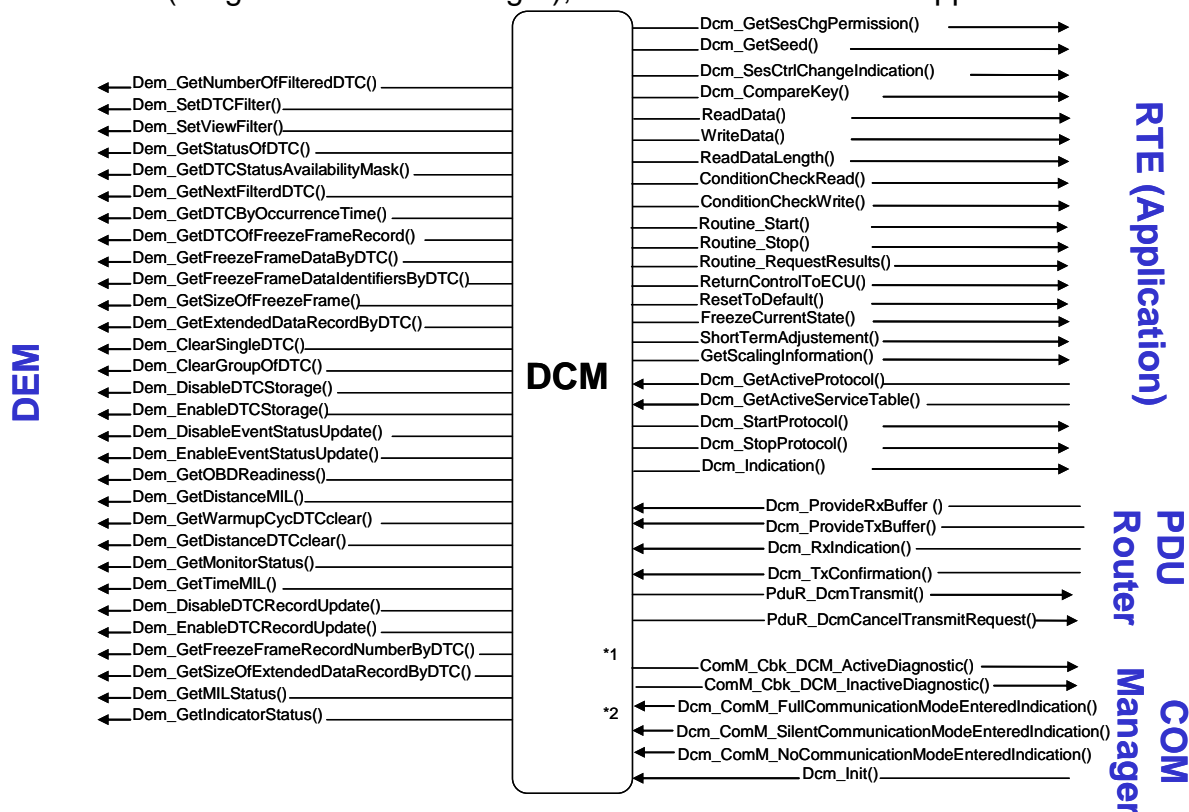
Each Parameter shall have a type. Types shall be based on primitive or complex AUTOSAR Types (i.e. they may be combined to structures, arrays etc.).

Parameters based on a “define” statement shall be put down in a way that the type can be checked by tool.

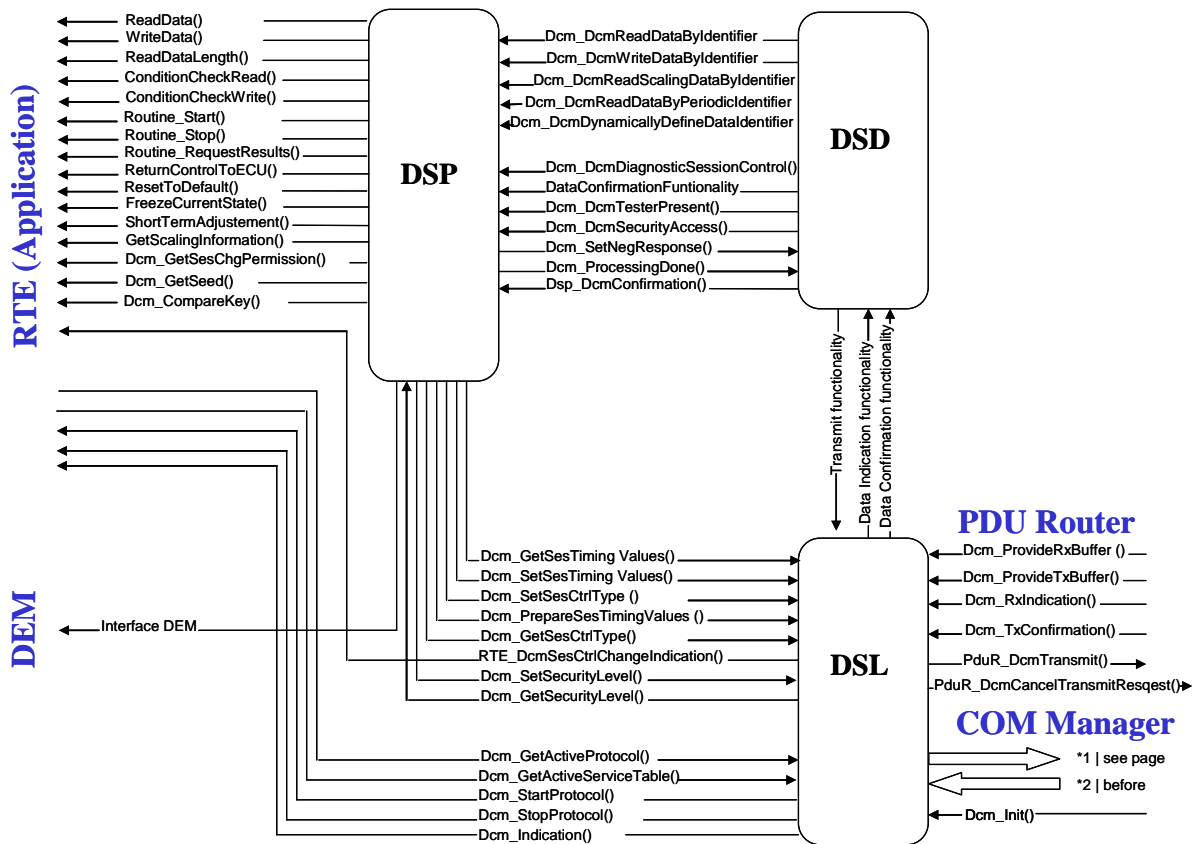
Dcm062: Each parameter shall have a list of valid values or the minimum as well as maximum values shall be specified.

8.1.1 Interface description of the DCM Module

The graphic below shows the interface between DCM and the other upper or lower Modules. The description of the Interface shall give a simple overview of the relation to the DEM (Diagnostic Event Manager), PDU Router and to the Application.



Additional the next graphic below gives the interface description between DSP, DSD and DSL in the DCM Module to have a quick overview of the relation inside the DCM.



8.2 Imported types

8.2.1 Standard types

In this chapter all included types are listed:

Used standard types of ComStack_Types.h:

- PduInfoType
- PduLengthType
- PduIdType
- BufReq_ReturnType
- NotifResultType

Used standard types of Platform_Types.h:

- uint8
- uint16
- boolean

Other used standard types

- Std_VersionInfoType

8.2.2 Other types

The DCM doesn't use types included from other modules.

8.3 Type definitions

Dcm063:

8.3.1 DCM Types

8.3.1.1 Dcm_StatusType

Type:	typedef		
Range:	Std_ReturnType	DCM_E_OK	0 : E_OK This value is representing a successful operation.
		DCM_E_NOT_OK	1 : E_NOT_OK This value is representing a unsuccessful operation.
		DCM_E_TI_PREPARE_LIMITS	2 New timing parameter are not ok, since requested values are not within the defined limits (used at API: Dcm_PrepareSesTiming-Values())
		DCM_E_TI_PREPARE_INCONSTENT	3 New timing parameter are not ok, since requested values are not consistent (e.g. P2min not smaller than P2max) (used at API: Dcm_PrepareSesTiming-Values ())
		DCM_E_SESSION_NOT_ALLOWED	4 Application does not allow start of requested session
		DCM_E_PROTOCOL_NOT_ALLOWED	5 Application does not allow start of requested protocol
		DCM_E_ROE_NOT_ACCEPTED	6 ResponseOnOneEvent request is not accepted by DCM (e.g. old ResponseOnOneEvent is not finished)
		DCM_E_PERIODICIDENT_NOACCEPTED	7 Periodic transmission request is not accepted by DCM (e.g. old Periodic transmission is not finished)
		DCM_E_REQUEST_NOT_ACCEPTED	8 Application rejects diagnostic request ->

		DCM_E_REQUEST_ENV_N OK	9 Diagnostic request is not allowed by application because of not fitting environmental conditions ->
		DCM_E_PENDING:	10 application process not yet completet, another call of application is required (this allows cyclic processing of application) (used at API: Dcm_GetSesChgPermission())
		DCM_E_COMPARE_KEY_F AILED	11 ECU compares by tester requested key with own calculated key When comparison fails this definition is used.
		DCM_E_FORCE_RCRRP	12 Application request the transmission of a diagnostic response <i>Response Pending</i> (NRC 0x78) (used at API: Dcm_GetSesChgPermission())
Description:		Base item type to transport status information.	

8.3.1.2 Dcm_ProtocolType

Type:	typedef		
Range:	uint8	0x00	DCM_OBD_ON_OBD on CAN (ISO15765-4; ISO15031-5) _CAN
		0x01	DCM_UDS_ON_UDS on CAN (ISO15765-3; ISO14229-1) _CAN
		0x02	DCM_UDS_ON_UDS on FlexRay (Manufacturer specific; _FLEXRAY ISO14229-1)
		0x03	DCM_ROE_ON_Response On Event on CAN _CAN
		0x04	DCM_ROE_ON_Response On Event on FlexRay _FLEXRAY
		0x05	DCM_PERIOD_Periodic Transmission on CAN ICTRANS_ON _CAN
		0x06	DCM_PERIOD_Periodic Transmission on FlexRay ICTRANS_ON _FLEXRAY
		0x07.. 0xEF	Reserved for further AUTOSAR implementation
	0xF0.. 0xFF	Reserved for SW supplier specific	
Description:		Protocol type definition	

8.3.1.3 Dcm_MsgItemtype

Type:	typedef
Range:	uint8

Description:	Base type for diagnostic message item
---------------------	---------------------------------------

8.3.1.4 Dcm_MsgType

Type:	typedef
Range:	Dcm_MsgItemType*
Description:	Base type for diagnostic message (request, positive or negative response)

8.3.1.5 Dcm_MsgLenType

Type:	typedef
Range:	uint16
Description:	Length of diagnostic message (request, positive or negative response). The maximum length is dependent of the underlying transport protocol/media. E. g. the maximum message length for CAN Transport Layer is 4095bytes.

8.3.1.6 Dcm_MsgAddInfoType

Please note that the following table describes a struct type definition - including its struct items "elements".

Type:	struct typedef
Range:	uint8
Description:	Additional information on message request.

```

Bit0 reqType(0 = physical request;
      1 = functional request)
Bit1 suppressPosResponse(0 = no (do not
                          suppress);
                          1 = yes (no positive response will
                          be sent))
Bit2 --
Bit3 --
Bit4 --
Bit5 --
Bit6 --
Bit7 --
    
```

8.3.1.7 Dcm_IdContextType

Type:	typedef
Range:	uint8
Description:	This message context identifier can be used to determine the relation between request and response confirmation.

8.3.1.8 Dcm_MsgContextType

Please note that the following table describes a struct type definition - including its struct items "elements".

Type:	struct typedef	
Element:	Dcm_MsgType	reqData: Request data, starting directly after service identifier (which is not part of this data)
Element:	Dcm_MsgLenType	reqDataLen: Request data length (excluding service identifier)
Element:	Dcm_MsgType	resData: Positive response data, starting directly after service identifier (which is not part of this data). The service identifier will be determined by DCM itself.
Element:	Dcm_MsgLenType	resDataLen: Positive response data length (excluding service identifier)
Element:	Dcm_MsgAddInfoType	msgAddInfo: Additional information about service request and response (see: Dcm_MsgAddInfo)
Element:	Dcm_MsgLenType	resMaxDataLen: The maximal length of a response is restricted by the size of the buffer. The buffer size can depend on the diagnostic protocol identifier which is assigned to this message, e. g. an OBD protocol id can obtain other properties than the enhanced diagnostic protocol id. The resMaxDataLen is a property of the diagnostic protocol assigned by the DSL. The value does not change during communication. It cannot be implemented as a constant, because it can differ between different diagnostic protocols.
Element	Dcm_IdContextType	idContext: This message context identifier can be used to determine the relation between request and response confirmation. This identifier can be stored within the application at request time, so that the response can be assigned to the original request. Background: Within the confirmation, the message context is no more valid, all message data is lost. You need an additional information to determine the request to which this confirmation belongs.
Element	PduldType	dcmRxDuld: Pdu identifier on which the request was received. The Pduld of the request can have consequences for message processing. E. g. an OBD request will be received on the OBD Pduld and will be processed slightly different than an enhanced diagnostic request received on the physical Pduld.
Description:	This data structure contains all information which is necessary to process a diagnostic message from request to response and response confirmation.	

8.3.1.9 Dcm_NegativeResponseCodeType

For the implementation of this table a comment or a special concept is needed in the c-code because the table is not structured conform to the MISRA rules.

Type:	typedef			
Range:	uint8	0x10	DCM_E_GENERALREJECT	This return value indicates that the requested action has been rejected by the application. The generalReject return value shall only be implemented in the application if none of the negative return values defined in this document meet the needs of the implementation. At no means shall this return value be a general replacement for the return values defined in this document.
		0x11	DCM_E_SERVICE_UNSUPPORTED	This return value indicates that the requested action will not be taken because the application does not support the requested service. The application shall send this return value in case the client has sent a request message with a service identifier, which is either unknown or not supported by the application. Therefore this negative return value is not shown in the list of negative return values to be supported for a diagnostic service, because this negative return value is not applicable for supported services.
		0x12	DCM_E_SUBFUNCTION_UNSUPPORTED	This return value indicates that the requested action will not be taken because the application does not support the service specific parameters of the request message. The application shall send this return value in case the client has sent a request message with a known and supported service identifier but with "sub function" which is either unknown or not supported.
		0x13	DCM_E_INCORRECTMESSAGELENGTHORINVALIDFORMAT	This return value indicates that the requested action will not be taken because the length of the received request message does not match the prescribed length for the specified service or the format of the parameters do not match the prescribed format for the specified service.
		0x14	DCM_E_RESPONSE_TOO_LONG	This return value indicates that the buffer of the lower layer is not large enough to transmit all data of the response

	0x21	DCM_E_BUSYR EPEATREQUES T	<p>This return value indicates that the application is temporarily too busy to perform the requested operation. In this circumstance the client shall perform repetition of the "identical request message" or "another request message". The repetition of the request shall be delayed by a time specified in the respective implementation documents.</p> <p>Example: In a multi-client environment the diagnostic request of one client might be blocked temporarily by a NRC 0x21 while a different client finishes a diagnostic task.</p> <p>Note: If the application is able to perform the diagnostic task but needs additional time to finish the task and prepare the response, the NRC 0x78 shall be used instead of NRC 0x21.</p> <p>This return value is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable return values of the diagnostic services.</p>
	0x22	DCM_E_CONDI TIONSNOTCORR ECT	<p>This return value indicates that the requested action will not be taken because the application prerequisite conditions are not met.</p>
	0x24	DCM_E_REQUE STSEQUENCEE RROR	<p>This return value indicates that the requested action will not be taken because the application expects a different sequence of request messages or message as sent by the client. This may occur when sequence sensitive requests are issued in the wrong order.</p> <p>EXAMPLE: A successful SecurityAccess service specifies a sequence of requestSeed and sendKey as sub-functions in the request messages. If the sequence is sent different by the client the application shall send a negative response message with the negative return value 0x24- requestSequenceError.</p>
	0x31	DCM_E_REQUE STOUTOFRANG E	<p>This return value indicates that the requested action will not be taken because the application has detected that the request message contains a parameter which attempts to substitute a value beyond its range of authority (e.g. attempting to substitute a data byte of 111 when the data is only defined to 100), or which attempts to access a dataIdentifier_routineIdentifier that is not supported or not supported in active session.</p> <p>This return value shall be implemented for all services, which allow the client to read data, write data or adjust functions by data in the application.</p>

	0x33	DCM_E_SECURITYACCESSDENIED	<p>This return value indicates that the requested action will not be taken because the application's security strategy has not been satisfied by the client.</p> <p>The application shall send this return value if one of the following cases occur:</p> <ul style="list-style-type: none"> - the test conditions of the application are not met, - the required message sequence e.g. DiagnosticSessionControl, securityAccess is not met, - the client has sent a request message which requires an unlocked application. <p>Beside the mandatory use of this negative return value as specified in the applicable services within this standard, this negative return value can also be used for any case where security is required and is not yet granted to perform the required service.</p>
	0x35	DCM_E_INVALIDKEY	<p>This return value indicates that the application has not given security access because the key sent by the client did not match with the key in the application's memory. This counts as an attempt to gain security. The application shall remain locked and increment is internal securityAccessFailed counter.</p>
	0x36	DCM_E_EXCEEDNUMBEROFATTEMPTS	<p>This return value indicates that the requested action will not be taken because the client has unsuccessfully attempted to gain security access more times than the application's security strategy will allow.</p>
	0x37	DCM_E_REQUIREDTIMEDELAYNOTEXPIRED	<p>This return value indicates that the requested action will not be taken because the client's latest attempt to gain security access was initiated before the application's required timeout period had elapsed.</p>
	0x70	DCM_E_UPLOADDOWNLOADNOTACCEPTED	<p>This return value indicates that an attempt to upload_download to a application's memory cannot be accomplished due to some fault conditions.</p>
	0x71	DCM_E_TRANSFERDATASUSPENDED	<p>This return value indicates that a data transfer operation was halted due to some fault.</p>
	0x72	DCM_E_GENERALPROGRAMMINGFAILURE	<p>This return value indicates that the application detected an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory).</p>
	0x73	DCM_E_WRONGBLOCKSEQUENCECOUNTER	<p>This return value indicates that the application detected an error in the sequence of blockSequenceCounter values. Note that the repetition of a TransferData request message with a blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the application.</p>

	0x7E	DCM_E_SUBFUNCTIONNOTSUPPORTEDINACTIVESESSION	This return value indicates that the requested action will not be taken because the application does not support the requested sub-function in the session currently active. This return value shall only be used when the requested sub-function is known to be supported in another session, otherwise return value SFNS (subFunctionNotSupported) shall be used. This return value shall be supported by each diagnostic service with a sub-function parameter, if not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable return values of the diagnostic services.
	0x7F	DCM_E_SERVICENOTSUPPORTEDINACTIVESESSION	This return value indicates that the requested action will not be taken because the application does not support the requested service in the session currently active. This return value shall only be used when the requested service is known to be supported in another session, otherwise return value SNS (serviceNotSupported) shall be used. This return value is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable return values of the diagnostic services.
	0x81	DCM_E_RPMTOO HIGH	This return value indicates that the requested action will not be taken because the application prerequisite condition for RPM is not met (current RPM is above a pre-programmed maximum threshold).
	0x82	DCM_E_RPMTOO LOW	This return value indicates that the requested action will not be taken because the application prerequisite condition for RPM is not met (current RPM is below a pre-programmed minimum threshold).
	0x83	DCM_E_ENGINEISRUNNING	This is required for those actuator tests which cannot be actuated while the Engine is running. This is different from RPM too high negative response, and needs to be allowed.
	0x84	DCM_E_ENGINEISNOTRUNNING	This is required for those actuator tests which cannot be actuated unless the Engine is running. This is different from RPM too low negative response, and needs to be allowed.
	0x85	DCM_E_ENGINERUNTIME TOO LOW	This return value indicates that the requested action will not be taken because the application prerequisite condition for engine run time is not met (current engine run time is below a pre-programmed limit).

	0x86	DCM_E_TEMPERATURETOOHIGH	This return value indicates that the requested action will not be taken because the application prerequisite condition for temperature is not met (current temperature is above a pre-programmed maximum threshold).
	0x87	DCM_E_TEMPERATURETOOLOW	This return value indicates that the requested action will not be taken because the application prerequisite condition for temperature is not met (current temperature is below a pre-programmed minimum threshold).
	0x88	DCM_E_VEICLESPEEDTOOHIGH	This return value indicates that the requested action will not be taken because the application prerequisite condition for vehicle speed is not met (current VS is above a pre-programmed maximum threshold).
	0x89	DCM_E_VEICLESPEEDTOOLOW	This return value indicates that the requested action will not be taken because the application prerequisite condition for vehicle speed is not met (current VS is below a pre-programmed minimum threshold).
	0x8A	DCM_E_THROTTLE_PEDALTOOHIGH	This return value indicates that the requested action will not be taken because the application prerequisite condition for throttle/pedal position is not met (current TP/APP is above a pre-programmed maximum threshold).
	0x8B	DCM_E_THROTTLE_PEDALTOOLOW	This return value indicates that the requested action will not be taken because the application prerequisite condition for throttle/pedal position is not met (current TP/APP is below a pre-programmed minimum threshold).
	0x8C	DCM_E_TRANSMISSIONRANGE NOTINNEUTRAL	This return value indicates that the requested action will not be taken because the application prerequisite condition for being in neutral is not met (current transmission range is not in neutral).
	0x8D	DCM_E_TRANSMISSIONRANGE NOTINGEAR	This return value indicates that the requested action will not be taken because the application prerequisite condition for being in gear is not met (current transmission range is not in gear).
	0x8F	DCM_E_BRAKESWITCH_NOTCLOSED	For safety reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test.
	0x90	DCM_E_SHIFTLEVERNOTINPARK	For safety reasons, this is required for certain tests before it begins, and must be maintained for the entire duration of the test.

	0x91	DCM_E_TORQUECONVERTERCLUTCHLOCKED	This return value indicates that the requested action will not be taken because the application prerequisite condition for torque converter clutch is not met (current TCC status above a pre-programmed limit or locked).
	0x92	DCM_E_VOLTAGE_TOO_HIGH	This return value indicates that the requested action will not be taken because the application prerequisite condition for voltage at the primary pin of the application (ECU) is not met (current voltage is above a pre-programmed maximum threshold).
	0x93	DCM_E_VOLTAGE_TOO_LOW	This return value indicates that the requested action will not be taken because the application prerequisite condition for voltage at the primary pin of the application (ECU) is not met (current voltage is below a pre-programmed maximum threshold).
Description:	This Table of available Negative Response Codes represents the allowed Response Codes an AUTOSAR SW Component shall return after a function call. For the allowed NRC of the executed Service ID please refer to the specification of the service in ISO14229-1 (UDS) and ISO15031-5 (OBD/CARB) (see chapter 4.2.4 Response code parameter definition Table 12).		

Table 4: Table of available Negative Response Codes

NOTE: The range of value 0x38 to 0x4F is reserved by ISO 15764 Extended data link security.

8.3.2 DSL (Diagnostic Session Layer) Types

8.3.2.1 Dcm_TimerServerType

PLEASE NOTE THAT THE FOLLOWING TABLE DESCRIBES A STRUCT TYPE DEFINITION - INCLUDING ITS STRUCT ITEMS "ELEMENTS".

Type:	struct typedef
Element:	uint16 Timer_P2ServerMin
Element:	uint16 Timer_P2ServerMax
Element:	uint16 Timer_P2StarServerMin
Element:	uint16 Timer_P2StarServerMax
Element:	uint16 Timer_S3Server
Description:	This data structure contains all timing parameters, valid for a diagnostic protocol. With reference to [8], following relation exists P2 _{CAN_Server} min value-> Timer_P2ServerMin P2 _{CAN_Server} max value -> Timer_P2ServerMax

	P2* _{CAN_Server} min value -> Timer_P2StarServerMin P2* _{CAN_Server} max value-> Timer_P2StarServerMax S3 _{Server} -> Timer_ S3Server Resolution for all elements is in ms. This resolution is not in accordance to the resolution of the service "AccessTimingParameter" This type is only used if the service AccessTimingParameter is used.
--	--

8.3.2.2 Dcm_TimerModeType

Type:	enumeration typedef		
Range:	0x00	DCM_DEFAULT	Configured Default timing parameters
	0x01	DCM_CURRENT	Current active timing parameters
	0x02	DCM_LIMITS	Configured Timing parameters limits
Description:	Different types of timing parameters This type is used for the API Dcm_GetSesTimingValues() to get values of configured default timing parameters, values of current active timing parameters or values of configured timing parameter limits.		

8.3.2.3 Dcm_SesCtrlType

Type:	typedef		
Range:	uint8	0x01	DEFAULT_SESSION
		0x02	PROGRAMMING_SESSION
		0x03	EXTENDED_DIAGNOSTIC_SESSION
		0x04	SAFETY_SYSTEM_DIAGNOSTIC_SESSION
		0x05...0x7F	<configuration dependent (according to "diagnosticSessionType" parameter DiagnosticSessionControl request (see [11]))>
		0x80...0xFE	Reserved by Document
		0xFF	ALL_SESSION_LEVEL
Description:	Session type definition		

8.3.2.4 Dcm_SecLevelType

Type:	typedef		
Range:	uint8	0x00	DCM_SEC_LEV_LOCKED
		0x01	DCM_SEC_LEV_L1
		0x02...0x7F	<configuration dependent>
		0x80...0xFE	Reserved by Document
		0xFF	DCM_SEC_LEV_ALL
Description:	Security Leveltype definition		

8.3.2.5 Dcm_ConfirmationStatusType

Type:	typedef		
Element:	uint8	0	DCM_RES_POS_OK
Element:		1	DCM_RES_POS_NOT_OK

Element:		2	DCM_RES_NEG_OK
Element:		3	DCM_RES_NEG_NOT_OK
Description:	Additional information on message request.		

8.4 Function definitions

This is a list of functions provided for other modules.

8.4.1 DCM Functions

8.4.1.1 Dcm_Init

Dcm037:

Service name:	Dcm_Init
Syntax:	void Dcm_Init (void)
Service ID:	0x01
Sync/Async:	Synchronous
Reentrancy:	Not Reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	Service for basic initialization of DCM module. The Initialization function will initialize all DCM global variables with the values of the configuration (see chapter 10 Configuration specification)
Caveats:	The call of this service is mandatory before using the DCM module for further processing.
Configuration:	None

8.4.1.2 Dcm_GetVersionInfo

Dcm065:

Service name:	Dcm_GetVersionInfo
Syntax:	void Dcm_GetVersionInfo (Std_VersionInfoType *versioninfo)
Service ID [hex]:	0x24
Sync/Async:	Synchronous
Reentrancy:	non reentrant
Parameters (in):	none
Parameters (out):	versioninfo Pointer to where to store the version information of this module.
Return value:	none

Description:	<p>This service returns the version information of this module. The version information includes:</p> <ul style="list-style-type: none"> - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). <p>This function shall be pre compile time configurable On/Off by the configuration parameter: DCM_VERSION_INFO_API</p> <p>Hint: If source code for caller of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.</p>
Caveats:	None
Configuration:	DcmVersionInfoApi and configuration parameters in chapter 10.6

8.4.2 DSL (Diagnostic Session Layer) Functions

Security Level Access

8.4.2.1 Dcm_GetSecurityLevel

Service name:	Dcm_GetSecurityLevel
Syntax:	Std_ReturnType Dcm_GetSecurityLevel (Dcm_SecLevelType * SecLevel)
Service ID:	0x0D
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (out):	<p>SecLevel</p> <p>Active Security Level value Conversion formula to calculate SecurityLevel out of tester requested SecurityAccessType parameter: $SecurityLevel = (SecurityAccessType + 1) / 2$ Content of SecurityAccessType is according to "securityAccessType" parameter of SecurityAccess request (see [11])</p>
Return value:	DCM_E_OK
Description:	DCM provides with this the active security level value This function can be called by the Application and DCM internal
Caveats:	None
Configuration:	None

Session control

8.4.2.2 Dcm_GetSesCtrlType

Service name:	Dcm_GetSesCtrlType
Syntax:	Std_ReturnType Dcm_GetSesCtrlType (Dcm_SesCtrlType * SesCtrlType)
Service ID:	0x06
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (out):	SesCtrlType Active Session Control Type value Content is according to "diagnosticSessionType" parameter of DiagnosticSessionControl request (see [11])
Return value:	DCM_E_OK
Description:	DCM provides with this the active session control type value
Caveats:	None
Configuration:	None

Protocol information

8.4.2.3 Dcm_GetActiveProtocol

Service name:	Dcm_GetActiveProtocol
Syntax:	Std_ReturnType Dcm_GetActiveProtocol(Dcm_ProtocolType * ActiveProtocol)
Service ID:	0x0F
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (out):	ActiveProtocol Active diagnostic protocol id (OBD, ENHANCED,...)
Return value:	DCM_E_OK
Description:	DCM provides with this the active Protocol name. This function can be called by the Application and DCM internal
Caveats:	None
Configuration:	DcmDslProtocolID

8.4.3 DSD (Diagnostic Service Dispatcher)

8.5 Call-back notifications

The following is a list of functions provided for lower layer modules. The function prototypes of the callback functions will be provided in the file Dcm_Cbk.h

8.5.1 DSL (Diagnostic Session Layer) Functions

Interface for PDU Router

Type definition for PduIdType, PduLengthType, PduInfoType, NotifResultType and BufReq_ReturnType can be found in the interface description of PDU Router (see [1]).

8.5.1.1 Dcm_ProvideRxBuffer

Service name:	Dcm_ProvideRxBuffer	
Syntax:	<pre>BufReq_ReturnType Dcm_ProvideRxBuffer (PduIdType DcmRxPduId, PduLengthType TpSduLength, PduInfoType** PduInfoPtr)</pre>	
Service ID:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	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	This length identifies the overall number of bytes to be received. The length shall be greater than zero.
Parameters (out):	PduInfoPtr	Pointer to pointer to PduInfoStructure containing data pointer and length of a receive buffer.
Return value:	BUFREQ_OK	Buffer request accomplished successful.
	BUFREQ_E_NOT_OK	Buffer request not successful. Buffer cannot be accessed by TP.
	BUFREQ_E_BUSY	No buffer is available at the moment.
	BUFREQ_E_OVFL	DCM is not capable to receive the number of TpSduLength Bytes.
Description:	<p>By this service the DCM is requested to provide a buffer for a transport layer on first frame or single frame reception.</p> <p>Within this function the DCM shall provide a pointer to a PduInfoStructure that contains a pointer to an SDU buffer and the length of this buffer. This information shall be passed via the output parameter <code>**PduInfoPtr</code>.</p> <p>By this service the receiver (e.g. DCM) is also informed implicitly about a first frame reception or a single frame reception.</p> <p>If a return value not equal to <code>BUFREQ_E_OK</code> is returned, the values of the out Parameters are not specified and shall not be evaluated.</p>	
Caveats:	<p>After returning a valid buffer, the DCM must not access this buffer unless:</p> <ul style="list-style-type: none"> it is being notified by the service <code>Dcm_RxIndication</code> about the successful reception (indication), or it is being notified by the service <code>Dcm_RxIndication</code> the reception was aborted (error indication). <p>The transport protocol filling the provided buffer will also set the length information contained in <code>PduInfoPtr</code> to the number of bytes that are valid in</p>	

	this buffer.
Configuration:	None

8.5.1.2 Dcm_RxIndication

Service name:	Dcm_RxIndication	
Syntax:	<pre>void Dcm_RxIndication (PduIdType DcmRxPduId, NotifResultType Result)</pre>	
Service ID:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	re-entrant	
Parameters (in):	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	Result of the N-PDU reception: <ul style="list-style-type: none"> • NTFRSLT_OK if the complete N-PDU has been received. • NTFRSLT_E_NOT_OK if an error occurred during reception, used to enable unlocking of the receive buffer.
Parameters (out):	None	
Return value:	None	
Description:	This function is called by the lower layer (in general the PDU Router): <ul style="list-style-type: none"> • with Result = NTFRSLT_OK after the complete DCM I-PDU has successfully been received, i.e. at the very end of the segmented TP receive cycle or after receiving an unsegmented N-PDU. • with Result = NTFRSLT_E_NOT_OK it is indicated that an error (e.g. timeout) has occurred during the reception of the DCM I-PDU. This passes the receive buffer back to DCM and allows error handling. It is undefined which part of the buffer contains valid data in this case, so the DCM shall not evaluate that buffer. By calling this service only the DCM is allowed to access the buffer. <Further detailed description>	
Caveats:	This function might be called in interrupt context. If an existing reception has to be canceled to establish a new reception it is required to indicate a cancellation first before requesting a buffer for the new reception. Otherwise DCM will be requested to open a second connection.	
Configuration:	--	

8.5.1.3 Dcm_ProvideTxBuffer

Service name:	Dcm_ProvideTxBuffer
Syntax:	<pre>BufReq_ReturnType Dcm_ProvideTxBuffer (PduIdType DcmTxPduId, PduInfoType**PduInfoPtr,</pre>

	uint16 Length)	
Service ID:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	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 <code>PduR_DcmTransmit()</code> .
	Length	<p>This is the minimum length given in bytes of the buffer requested from the DCM. The minimum length is needed by the transport protocol to perform error recovery mechanisms.</p> <p>The value of this parameter shall not exceed the number of Bytes still to be sent.</p> <p>A length of zero indicates that the length of the buffer can be of arbitrary size (larger than zero).</p> <p>The Length parameter is expected by DCM to be always 0 (zero).</p>
Parameters (out):	PduInfoPtr	<p>Pointer to pointer to PduInfoStructure containing data pointer and length of a transmit buffer.</p> <p>This length must not be smaller than the length given by <code>MinLength</code>.</p>
Return value:	BUFREQ_OK	Buffer request accomplished successful.
	BUFREQ_E_NOT_OK	Buffer request not successful. Buffer cannot be accessed by TP.
	BUFREQ_E_BUSY	DCM temporarily cannot provide a buffer of the requested length.
Description:	<p>By this service the DCM is requested to provide a buffer containing data to be transmitted via a transport protocol.</p> <p>The length of the buffer does not need to be in the length of the DCM SDU to be transmitted. It only needs to be as large as required by the caller of that service (<code>Length</code>).</p> <p>If the returned buffer is smaller than the length requested to transmit within the service <code>PduR_DcmTransmit()</code> the DCM will be requested by this service to provide another buffer after the data of the current buffer have been transmitted.</p> <p>If the DCM can provide a buffer of the size <code>Length</code> or larger, this service returns <code>BUFREQ_OK</code>.</p> <p>If the DCM cannot provide a buffer of the size <code>Length</code>, or larger (except <code>Length</code> is equal zero) an the service returns with <code>BUFREQ_E_NOT_OK</code>.</p> <p>If the DCM temporarily cannot provide a buffer of the requested length, the service returns with <code>BUFREQ_E_BUSY</code>. This service has to be called again during next processing of the <code>MainFunction</code> of the transport protocol</p>	
Caveats:	<p>After returning a valid buffer, the DCM must not access this buffer unless it is requested to provide a new buffer by this service for the same <code>DcmTxPduId</code> or it is notified about the successful transmission (confirmation) or it is notified by an error indicating that the transmission was aborted (error indication).</p> <p>If this service returns <code>BUFREQ_E_NOT_OK</code> the transmit requests issued by calling the service <code>PduR_DcmTransmit</code> is still not finished. A final confirmation (indicating an error) is required to finish this service and to start a subsequent request by calling <code>PduR_DcmTransmit</code>. So it is up to the transport protocol if the transmission is aborted on <code>BUFREQ_E_NOT_OK</code> or not.</p>	
Configuration:	None	

8.5.1.4 Dcm_TxConfirmation

Service name:	Dcm_TxConfirmation	
Syntax:	<pre>void Dcm_TxConfirmation (PduIdType DcmTxPduId, NotifResultType Result)</pre>	
Service ID:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	DcmTxPduId	ID of DCM I-PDU that has been transmitted. Range: 0..(maximum number of I-PDU IDs transmitted by DCM) – 1
	Result	Result of the N-PDU transmission: <ul style="list-style-type: none"> • NTFRSLT_OK if the complete N-PDU has been transmitted. • NTFRSLT_E_NOT_OK if an error occurred during transmission, used to enable unlocking of the transmit buffer. • NTFRSLT_E_CANCELATION_OK if the N-PDU has been successfully cancelled • NTFRSLT_E_CANCELATION_NOT_OK if an error occurred when cancelling the N-PDU
Parameters (out):	None	
Return value:	None	
Description:	This function is called by the lower layer (in general the PDU Router): <ul style="list-style-type: none"> • with Result = NTFRSLT_OK after the complete DCM I-PDU has successfully been transmitted, i.e. at the very end of the segmented TP transmit cycle. Within this function, the DCM shall unlock the transmit buffer. • with Result = NTFRSLT_E_NOT_OK if an error (e.g. timeout) has occurred during the transmission of the DCM I-PDU. This enables unlocking of the transmit buffer and error handling. • with Result = NTFRSLT_E_CANCELATION_OK if the N-PDU has been successfully cancelled after a request with PduR_DcmCancelTransmitRequest() • with Result = NTFRSLT_E_CANCELATION_NOT_OK if an error occurred when cancelling the N-PDU after a request with PduR_DcmCancelTransmitRequest() 	
Caveats:	This function might be called in interrupt context (e.g. from a transmit interrupt). However, for transmission via FlexRay there is a restriction: Since the FlexRay Specification does not mandate the existence of a transmit interrupt, the exact meaning of this confirmation (i.e. “transfer into the FlexRay controller’s send buffer” OR “transmission onto the FlexRay network”) depends on the capabilities of the FlexRay communication controller and the configuration of the FlexRay Interface.	
Configuration:	Only those I-PDUs are confirmed where a transmit confirmation is enabled by configuration.	

Interface for Communication Manager

8.5.1.5 Dcm_ComM_NoComModeEntered

Service name:	Dcm_ComM_NoComModeEntered
Syntax:	void Dcm_ComM_NoComModeEntered(void)
Service ID:	0x21
Sync/Async:	Synchronous
Reentrancy:	reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	All kind of transmissions (receive and transmit) shall be stopped immediately. This means that the ResponseOnEvent and PeriodicId and also the transmission of the normal communication (PduR_DcmTransmit) shall be disabled.
Caveats:	None
Configuration:	None

8.5.1.6 Dcm_ComM_SilentComModeEntered

Service name:	Dcm_ComM_SilentComModeEntered
Syntax:	void Dcm_ComM_SilentComModeEntered(void)
Service ID:	0x22
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	All outgoing transmissions shall be stopped immediately. This means that the ResponseOnEvent and PeriodicId and also the transmission of the normal communication (PduR_DcmTransmit) shall be disabled.
Caveats:	None
Configuration:	None

8.5.1.7 Dcm_ComM_FullComModeEntered

Service name:	Dcm_ComM_FullComModeEntered
Syntax:	void Dcm_ComM_FullComModeEntered(void)
Service ID:	0x23
Sync/Async:	Synchronous
Reentrancy:	reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	All kind of transmissions shall be enabled immediately. This means that the ResponseOnEvent and PeriodicId and also the transmission of the normal communication (PduR_DcmTransmit) shall be enabled.

Caveats:	None
Configuration:	None

8.6 Scheduled functions

8.6.1 Dcm_MainFunction

Dcm053:

Service name:	Dcm_MainFunction
Syntax:	void Dcm_MainFunction(void)
Service ID:	0x25
Sync/Async:	Synchronous
Reentrancy:	Non re-entrant
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	This service is used for processing the tasks of the main loop. It shall be called periodically as cyclic task by the software system (e.g. by operating system). In this function all Scheduled functions shall be integrated (e.g. Diagnostic timer handling)
Caveats:	The DCM Interface must be initialized.
Configuration:	DcmTaskTime

8.7 Expected Interfaces

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

8.7.1 Mandatory Interfaces

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

API function	Module	Description
<i>PduR_DcmTransmit</i>	<PduR>	forwards the diagnostic message to the transmitting layers (e.g. CAN TP)
<i>PduR_DcmCancelTransmitRequest</i>	<PduR>	Request the cancellation of a transmission
<i>Dem_GetNumberOfFilteredDTC</i>	<Dem>	DEM shall return the number of filtered DTCs (for the set filter)
<i>Dem_GetNextFilteredDTCAndFDC</i>	<Dem>	DEM shall return the current DTC and its associated Fault Detection Counter (FDC) from the DEM matching the filter criteria defined by the API call [7]
<i>Dem_SetDTCFilter</i>	<Dem>	DCM can set the DTC filter
<i>Dem_SetViewFilter</i>	<Dem>	DCM can set the VIEW filter
<i>Dem_GetStatusOfDTC</i>	<Dem>	DEM shall return the state of the

		requested DTC
<i>Dem_GetDTCStatusAvailabilityMask</i>	<Dem>	DEM shall return the StatusAvailabilityMask of the requested DTC
<i>Dem_GetNextFilterdDTC</i>	<Dem>	DEM shall return next filtered DTC (will be called in a loop)
<i>Dem_GetDTCByOccurrenceTime</i>	<Dem>	DEM shall return DTCs by occurrence time
<i>Dem_GetDTCOfFreezeFrameRecord</i>	<Dem>	DEM shall the DTC associated with the Freeze Frame selected via its absolute record number.
<i>Dem_GetSizeOfFreezeFrame</i>	<Dem>	DEM shall return the size of the requested FreezeFrame
<i>Dem_GetFreezeFrameDataByDTC</i>	<Dem>	DEM shall return FreezeFrame for requested DTCs
<i>Dem_GetFreezeFrameDataIdentifierByDTC</i>	<Dem>	DEM shall return FreezeFrame Data Identifiers for requested DTCs
<i>Dem_GetExtendedDataRecordByDTC</i>	<Dem>	DEM shall return extended data record for requested DTCs
<i>Dem_ClearDTC</i>	<Dem>	DEM shall erase requested DTCs
<i>Dem_DisableDTCStorage</i>	<Dem>	DEM shall disable the DTC storage
<i>Dem_EnableDTCStorage</i>	<Dem>	DEM shall enable the DTC storage
<i>Dem_DisableEventStatusUpdate</i>	<Dem>	DEM shall disable the Event Status Update
<i>Dem_EnableEventStatusUpdate</i>	<Dem>	DEM shall enable the Event Status Update
<i>Dem_DisableDTCRecordUpdate</i>	<Dem>	DEM shall disable the DTC Record Update
<i>Dem_EnableDTCRecordUpdate</i>	<Dem>	DEM shall enable the DTC Record Update
<i>Dem_GetSizeOfExtendedDataRecordByDTC</i>	<Dem>	DEM shall return size of extended data record for requested DTCs
<i>Dem_GetMILStatus</i>	<Dem>	DEM shall return MIL-Status required for ISO15031-5
<i>Dem_GetIndicatorStatus</i>	<Dem>	
<i>Dem_GetTimeDTCclear</i>	<Dem>	DEM shall return PID4Eh of ISO15031-5 Data A/B information
<i>ComM_DCM_ActiveDiagnostic</i>	<ComM>	With this function the DCM indicates the start of an Diagnostic communication
<i>ComM_DCM_InactiveDiagnostic</i>	<ComM>	With this function the DCM indicates the stop of an Diagnostic communication
<i>SchM_ActMainFunction_DCM</i>	<SchM>	The DCM module implementation invokes the SchM_ActMainFunction API to trigger the activation of a corresponding main processing function.
<i>SchM_CancelMainFunction_DCM</i>	<SchM>	The DCM module invokes the SchM_CancelMainFunction API to trigger the cancelation of the requested activation of a corresponding main processing function.
<i>Mcu_PerformReset</i>	<Mcu>	The DCM module invokes the Mcu_PerformReset API to reset the ECU.

8.7.2 Optional Interfaces

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

API function	Module	Description	Configuration parameter (description see chapter 10)
Det_ReportError	Det	Development error notification	DcmDevErrorDetect

8.7.3 RTE interfaces

In this chapter all interfaces are listed where the target function shall exist in the RTE. The target function is usually a call-back function.

8.7.3.1 DSL (Diagnostic Session Layer) Functions

8.7.3.1.1 Dcm_GetSesChgPermission

Service name:	Dcm_GetSesChgPermission		
Syntax:	<pre>Std_ReturnType Rte_Call_Dcm_SessionControl_GetSesChgPermission (Dcm_SesCtrlType SesCtrlTypeActive, Dcm_SesCtrlType SesCtrlTypeNew)</pre>		
Service ID:	0x08		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (in):	SesCtrlTypeActive	Active session control type value	
	SesCtrlTypeNew	Requested session control type value	
Parameters (out):	None		
Return value:	Permission of application. DCM_E_OK: session can be started DCM_E_SESSION_NOT_ALLOWED: session is not allowed to start DCM_E_PENDING: application process not yet completed, another call of Dcm_GetSesChgPermission() is required (this allows cyclic processing of Dcm_GetSesChgPermission()) DCM_E_FORCE_RCRRP : application request the transmission of a diagnostic response <i>Response Pending</i> (NRC 0x78)		
Description:	Request to application, to check if the conditions allows starting requested session control. The application needs to provide this function as a callback function. DCM provides only the prototype of this function.		
Caveats:	None		
Configuration:	None		

8.7.3.1.2 Dcm_SesCtrlChangeIndication

Service name:	Dcm_SesCtrlChangeIndication
Syntax:	Std_ReturnType Rte_Call_Dcm_SessionControl_SesCtrlChangeIndication (Dcm_SesCtrlType SesCtrlTypeOld, Dcm_SesCtrlType SesCtrlTypeNew)
Service ID:	0x09
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	SesCtrlTypeOld Former active session control type value SesCtrlTypeNew Newly set session control type value
Parameters (out):	None
Return value:	DCM_E_OK
Description:	Indication to application, that value of active session control is changed (including "self-transitions"). This change was either initiated by "DiagnosticSessionControl" command or S3Server timeout. The application needs to provide this function as a callback function. DCM provides only the prototype of this function.
Caveats:	None
Configuration:	None

8.7.3.1.3 Dcm_ConfirmationRespPend

Service name:	DCM_ConfirmationRespPend
Syntax:	Std_ReturnType Rte_Call_Dcm_SessionControl_ConfirmationRespPend (Dcm_ConfirmationStatusType status)
Service ID:	0x27
Sync/Async:	Synchronous
Reentrancy:	Not reentrant.
Parameters (in):	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.
Parameters (out):	None
Return value:	DCM_E_OK
Description:	This function confirms the successful transmission or a transmission error of a negative response with code 0x78, which was triggered directly by application (return value DCM_E_FORCE_RCRP on Dcm_GetSesChgPermission call)
Caveats:	None
Configuration:	DCM_FORCE_RESPPEND_ENABLED : Enable/Disable the presence of this API

8.7.3.1.4 Dcm_StartProtocol

Service name:	Dcm_StartProtocol
Syntax:	Std_ReturnType Rte_Call_Dcm_StartProtocol (Dcm_ProtocolType ProtocolID)
Service ID:	0x1F
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	ProtocolID Name of the protocol (IDs configured within DCM_PROTOCOL_ID)
Parameters (out):	None
Return value:	DCM_E_OK: conditions in application allows further processing of protocol DCM_E_PENDING: application process not yet completed, another call of Dcm_StartProtocol() is required (this allows cyclic processing of Dcm_StartProtocol()) DCM_E_PROTOCOL_NOT_ALLOWED: conditions in application allows no further procession of protocol
Description:	With this callback function, application is able to reject further processing of requested protocol due to failed conditions.
Caveats:	None
Configuration:	DcmDslProtocolID

8.7.3.1.5 Dcm_StopProtocol

Service name:	Dcm_StopProtocol
Syntax:	Std_ReturnType Rte_Call_Dcm_StopProtocol (Dcm_ProtocolType ProtocolID)
Service ID:	0x20
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	ProtocolID Name of the protocol (IDs configured within DCM_PROTOCOL_ID)
Parameters (out):	None
Return value:	DCM_E_OK
Description:	If a running diagnostic requested is preempted by a higher prior request (of an other protocol, e.g. OBD), application is requested to abort further processing of running request and finish with Dcm_ProcessingDone().
Caveats:	
Configuration:	None

8.7.3.1.6 Dcm_Indication

Service name:	Dcm_Indication
Syntax:	Std_ReturnType Rte_Call_Dcm_Indication (uint8 SID,)

	uint8 RequestData[<maximum data size>], uint16 DataSize, Dcm_NegativeResponseCodeType * ErrorCode)						
Service ID:	0x17						
Sync/Async:	Synchronous						
Reentrancy:	Reentrant						
Parameters (in):	<table border="1"> <tr> <td>SID</td> <td>Value of service identifier</td> </tr> <tr> <td>RequestData</td> <td>This parameter contains the complete request data (diagnostic buffer), except the service ID.</td> </tr> <tr> <td>DataSize</td> <td>This parameter defines how many bytes in the RequestData parameter are valid.</td> </tr> </table>	SID	Value of service identifier	RequestData	This parameter contains the complete request data (diagnostic buffer), except the service ID.	DataSize	This parameter defines how many bytes in the RequestData parameter are valid.
SID	Value of service identifier						
RequestData	This parameter contains the complete request data (diagnostic buffer), except the service ID.						
DataSize	This parameter defines how many bytes in the RequestData parameter are valid.						
Parameters (out):	ErrorCode						
Return value:	<p>DCM_E_OK: diagnostic request accepted</p> <p>DCM_E_REQUEST_NOT_ACCEPTED: application rejects diagnostic request -> DSD will ignore diagnostic request, no further processing of request, no response will be send</p> <p>DCM_E_REQUEST_ENV_NOK: Environmental conditions are not ok -> DSD will generate negative response with NRC 0x22 (ConditionsNotCorrect)</p>						
Description:	This function indicates the successful reception of a new request to application and it is called right before processing the diagnostic service is processed. Within this function application can examine the permission of the diagnostic service / environment (e.g. ECU state afterrun). Return of application will lead to an acceptance of diagnostic request, reject of diagnostic request (-> no response will be send) or to an NRC 0x22 (ConditionsNotCorrect)						
Caveats:							
Configuration:	DcmRequestIndicationEnabled						

8.7.3.2 Interface for special services

8.7.3.2.1 Dcm_GetSeed

Service name:	Dcm_GetSeed
Syntax:	Std_ReturnType Rte_Call_Dcm_SecurityAccess_<Level>_GetSeed (uint8 securityAccessDataRecord[<size>] uint8 Seed[<seed size>] Dcm_NegativeResponseCodeType * ErrorCode)
Service ID:	0x1D
Sync/Async:	Asynchronous
Reentrancy:	Non-Reentrant
Parameters (in):	securityAccessDataRecord Optional parameter: This data record contains – if present – additional data to calculate the seed value
Parameters (out):	Seed Pointer for provided seed ErrorCode
Return value:	<p>DCM_E_OK : key provided</p> <p>DCM_E_PENDING: application process not yet completed, another call of Dcm_GetSeed() is required (this allows cyclic processing of Dcm_GetSeed())</p>

Description:	Request to application for provision of seed value
Caveats:	None
Configuration:	DcmDspSecuritySeedSize

8.7.3.2.2 Dcm_CompareKey

Service name:	Dcm_CompareKey
Syntax:	<pre>Std_ReturnType Rte_Call_Dcm_SecurityAccess_<Level>_CompareKey (Uint8 Key[<key size>], Dcm_NegativeResponseCodeType * ErrorCode,) </pre>
Service ID:	0x1E
Sync/Async:	Asynchronous
Reentrancy:	Non-Reentrant
Parameters (in):	Key Key, which needs to be compared
Parameters (out):	ErrorCode
Return value:	DCM_E_OK: compare was successful DCM_E_PENDING: application process not yet completed, another call of Dcm_CompareKey() is required (this allows cyclic processing of Dcm_CompareKey()) DCM_E_COMPARE_KEY_FAILED: compare failed
Description:	Request to application for comparing key
Caveats:	None
Configuration:	DcmDspSecurityKeySize

8.7.4 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The full name of these interfaces is not fixed because parts of them are configurable.

8.7.4.1 Interfaces with RTE

8.7.4.1.1 ReadData

Service name:	ReadData
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_DidServices_<DID>_ReadData { <DID specific type>* data } </pre>
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	None
Parameters (out):	data Did value

Return value:	DCM_E_OK: Data has been successfully read DCM_E_PENDING: application process not yet completed, another call of ReadData is required (this allows cyclic processing of ReadData)
Description:	Request to application the data value of a DID. A ReadData interface is defined for each DID with read access The DID specific type is an array of uint8 which represent <ul style="list-style-type: none"> ➤ Either the fix length of this DID ➤ Or the maximum possible length of this DID. If the length is variable, the operation ReadDataLength has to provide the current valid data length of this DID. This interface is used for ReadDataByIdentifier and ReadDataByPeriodicIdentifier services
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.2 ReadDataLength

Service name:	ReadDataLength
Syntax:	Std_ReturnType Rte_Call_DCM_DidServices_<DID>_ReadDataLength { uint16 * DidLength }
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	None
Parameters (out):	DidLength Data length of the DID
Return value:	DCM_E_OK: Data length has been successfully read DCM_E_PENDING: application process not yet completed, another call of ReadDataLength is required (this allows cyclic processing of ReadDataLength)
Description:	Request to application the data length of a DID. A ReadDataLength interface is defined for each DID with variable data length This interface is used for ReadDataByIdentifier and ReadDataByPeriodicIdentifier services
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.3 WriteData

Service name:	WriteData
Syntax:	Std_ReturnType Rte_Call_DCM_DidServices_<DID>_WriteData { <DID specific type> data uint16 dataLength, Dcm_NegativeResponseCodeType * ErrorCode }
Service ID:	-
Sync/Async:	-

Reentrancy:	-	
Parameters (in):	data	Did value
	dataLength	Optional: Available, if the data length of the DID is variable. The parameter defines the size of valid data in the parameter data.
Parameters (out):	ErrorCode	Result : 0x00 : Conditions are correct 0x01...0xFF : NRC value corresponding to the non correctness cause
Return value:	DCM_E_OK: Data has been successfully written DCM_E_PENDING: application process not yet completed, another call of WriteData is required (this allows cyclic processing of WriteData)	
Description:	Request to application to write the data value of a DID. The DID specific type is an array of Uint8 which represent <ul style="list-style-type: none"> ➤ Either the fix length of this DID ➤ Or the maximum possible length of this DID. A WriteData interface is defined for each DID with write access This interface is used for WriteDataByIdentifier services	
Caveats:		
Configuration:	The CallbackRoutineName is configured...	

8.7.4.1.4 ConditionCheckRead

Service name:	ConditionCheckRead	
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_DidServices_<DID>_ConditionCheckRead { Dcm_NegativeResponseType * ErrorCode }</pre>	
Service ID:	-	
Sync/Async:	-	
Reentrancy:	-	
Parameters (in):	None	
Parameters (out):	ErrorCode	Result : 0x00 : Conditions are correct 0x01...0xFF : NRC value corresponding to the non correctness cause
Return value:	DCM_E_OK: Check has been done DCM_E_PENDING: application process not yet completed, another call of ConditionCheckRead is required (this allows cyclic processing of ConditionCheckRead)	
Description:	Request to application if the conditions (System state,...) to read the DID are correct. This operation is called for all requested DIDs before requesting the data of each DID. A ConditionCheckRead interface is defined for each DID with read access	
Caveats:		
Configuration:	The CallbackRoutineName is configured...	

8.7.4.1.5 ConditionCheckWrite

Service name:	ConditionCheckWrite						
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_DidServices_<DID>_ ConditionCheckWrite { Dcm_NegativeResponseCodeType * ErrorCode }</pre>						
Service ID:	-						
Sync/Async:	-						
Reentrancy:	-						
Parameters (in):	None						
Parameters (out):	<table> <tr> <td>ErrorCode</td> <td>Result :</td> </tr> <tr> <td></td> <td>0x00 : Conditions are correct</td> </tr> <tr> <td></td> <td>0x01...0xFF : NRC value corresponding to the non correctness cause</td> </tr> </table>	ErrorCode	Result :		0x00 : Conditions are correct		0x01...0xFF : NRC value corresponding to the non correctness cause
ErrorCode	Result :						
	0x00 : Conditions are correct						
	0x01...0xFF : NRC value corresponding to the non correctness cause						
Return value:	DCM_E_OK: Check has been done DCM_E_PENDING: application process not yet completed, another call of ConditionCheckWrite is required (this allows cyclic processing of ConditionCheckWrite)						
Description:	Request to application if the conditions (Security access, System state,...) to write the DID are correct A ConditionCheckWrite interface is defined for each DID with write access						
Caveats:							
Configuration:	The CallbackRoutineName is configured...						

8.7.4.1.6 Routine_Start

Service name:	Routine_Start						
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_RoutineServices_<RoutineName>_Start { <Routine specific parameters>, Dcm_NegativeResponseCodeType * ErrorCode }</pre>						
Service ID:	-						
Sync/Async:	-						
Reentrancy:	-						
Parameters (in):	<Routine specific parameters>						
Parameters (out):	<table> <tr> <td><Routine specific parameters></td> <td>Result :</td> </tr> <tr> <td>ErrorCode</td> <td>0x00 : Conditions are correct</td> </tr> <tr> <td></td> <td>0x01...0xFF : NRC value corresponding to the non correctness cause</td> </tr> </table>	<Routine specific parameters>	Result :	ErrorCode	0x00 : Conditions are correct		0x01...0xFF : NRC value corresponding to the non correctness cause
<Routine specific parameters>	Result :						
ErrorCode	0x00 : Conditions are correct						
	0x01...0xFF : NRC value corresponding to the non correctness cause						
Return value:	DCM_E_OK: Process has been completed DCM_E_PENDING: application process not yet completed, another call of Routine_Start is required (this allows cyclic processing of Routine_Start)						
Description:	Request to application to start a routine A Routine_Start interface is defined for each routine						

	This interface is used for RoutineControl service with sub function startRoutine.
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.7 Routine_Stop

Service name:	Routine_Stop
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_RoutineServices_<RoutineName>_Stop { <Routine specific parameters>, Dcm_NegativeResponseCodeType * ErrorCode }</pre>
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	<Routine specific parameters>
Parameters (out):	<pre><Routine specific parameters> ErrorCode</pre> <p>Result : 0x00 : Conditions are correct 0x01...0xFF : NRC value corresponding to the non correctness cause</p>
Return value:	DCM_E_OK: Process has been completed DCM_E_PENDING: application process not yet completed, another call of Routine_Stop is required (this allows cyclic processing of Routine_Stop)
Description:	Request to application to stop a routine A Routine_Stop interface is defined for each routine This interface is used for RoutineControl service with sub function stopRoutine.
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.8 Routine_RequestResults

Service name:	Routine_RequestResults
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_RoutineServices_<RoutineName>_RequestResult { <Routine specific parameters>, Dcm_NegativeResponseCodeType * ErrorCode }</pre>
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	-
Parameters (out):	<Routine specific parameters>

	<p>ErrorCode</p> <p>Result :</p> <p>0x00 : Conditions are correct</p> <p>0x01...0xFF : NRC value corresponding to the non correctness cause</p>
Return value:	<p>DCM_E_OK: Result has been provided</p> <p>DCM_E_PENDING: application process not yet completed, another call of Routine_RequestResults is required (this allows cyclic processing of Routine_RequestResults)</p>
Description:	<p>Request to application the results of a routine</p> <p>A Routine_RequestResults interface is defined for each routine</p> <p>This interface is used for RoutineControl service with sub function requestRoutineResults.</p>
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.9 ReturnControlToECU

Service name:	ReturnControlToECU
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_DidServices_<DID>_ReturnControlToEcu { <Control specific parameters> }</pre>
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	-
Parameters (out):	<Control specific parameters>
Return value:	<p>DCM_E_OK: Process has been completed</p> <p>DCM_E_PENDING: application process not yet completed, another call of ReturnControlToECU is required (this allows cyclic processing of ReturnControlToECU)</p>
Description:	<p>Request to application to return control to ECU of an IOControl</p> <p>A ReturnControlToECU interface is defined for each controllable DID</p> <p>This interface is used for InputOutputControlByIdentifier service service with controlOptionRecord returnControlToECU.</p>
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.10 ResetToDefault

Service name:	ResetToDefault
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_DidServices_<DID>_ResetToDefault { <Control specific parameters> }</pre>

	}
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	-
Parameters (out):	<Control specific parameters>
Return value:	DCM_E_OK: Process has been completed DCM_E_PENDING: application process not yet completed, another call of ResetToDefault is required (this allows cyclic processing of ResetToDefault)
Description:	Request to application to reset an IOControl to default value A ResetToDefault interface is defined for each controllable DID This interface is used for InputOutputControlByIdentifier service service with controlOptionRecord resetToDefault.
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.11 FreezeCurrentState

Service name:	FreezeCurrentState
Syntax:	Std_ReturnType Rte_Call_DCM_DidServices_<DID>_FreezeCurrentState { <Control specific parameters> }
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	-
Parameters (out):	<Control specific parameters>
Return value:	DCM_E_OK: Process has been completed DCM_E_PENDING: application process not yet completed, another call of FreezeCurrentState is required (this allows cyclic processing of FreezeCurrentState)
Description:	Request to application to freeze the current state of an IOControl A FreezeCurrentState interface is defined for each controllable DID This interface is used for InputOutputControlByIdentifier service service with controlOptionRecord freezeCurrentState.
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.12 ShortTermAdjustment

Service name:	ShortTermAdjustment
Syntax:	Std_ReturnType Rte_Call_DCM_DidServices_<DID>_ShortTermAdjustment { <Control specific parameters> }
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	<Control specific parameters>
Parameters (out):	<Control specific parameters>
Return value:	DCM_E_OK: Process has been completed DCM_E_PENDING: application process not yet completed, another call of ShortTermAdjustment is required (this allows cyclic processing of ShortTermAdjustment)
Description:	Request to application to adjuste the IO signal. A ShortTermAdjustment interface is defined for each controllable DID This interface is used for InputOutputControlByIdentifier service with controlOptionRecord shortTermAdjustment
Caveats:	
Configuration:	The CallbackRoutineName is configured...

8.7.4.1.13 GetScalingInformation

Service name:	GetScalingInformation
Syntax:	Std_ReturnType Rte_Call_DCM_DidServices_<DID>_GetScalingInformation { uint8 ScalingInfo[<Scaling Info size>] }
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	None
Parameters (out):	ScalingValue
Return value:	DCM_E_OK: Process has been completed DCM_E_PENDING: application process not yet completed, another call of GetScalingInformation is required (this allows cyclic processing of GetScalingInformation)
Description:	Request to application the scaling information of the DID This interface is used for ReadScalingDataByIdentifier services
Caveats:	

Configuration:	The CallbackRoutineName is configured...
-----------------------	--

8.7.4.1.14 EcuReset

Service name:	EcuReset
Syntax:	<pre>Std_ReturnType Rte_Call_DCM_ResetServices_EcuReset { uint8 resetType, Dcm_NegativeResponseCodeType * ErrorCode }</pre>
Service ID:	-
Sync/Async:	-
Reentrancy:	-
Parameters (in):	resetType Reset type as defined in ISO14229
Parameters (out):	ErrorCode Result : 0x00 : Conditions are correct 0x01...0xFF : NRC value corresponding to the non correctness cause
Return value:	DCM_E_OK: Process has been completed DCM_E_PENDING: application process not yet completed, another call of EcuReset is required (this allows cyclic processing of EcuReset)
Description:	Request to application to grant the execution of an ECU reset. This may also require to shutdown the application before granting the reset (e.g. storing all non volatile data) This interface is used for EcuReset services with sub function HardReset.
Caveats:	
Configuration:	

8.7.4.2 DSD (Diagnostic Service Dispatcher)

All ISO 14229-1 and 15031-5 protocol services are handled the same way. The service identifier is already checked and approved when those functions are called. This is the complete list of service callback function examples.

The function names shall be provided in the format `Dsp_DcmServiceName` for each supported diagnostic service identifier.

Service Identifier	Function name for <DiagnosticService>
0x10	Dsp_DcmDiagnosticSessionControl
0x11	Dsp_DcmECUReset
0x27	Dsp_DcmSecurityAccess
0x28	NRC "ServiceNotSupported" (Dsp_DcmCommunicationControl)
0x3E	Dsp_DcmTesterPresent
0x83	NRC "ServiceNotSupported" (Dsp_DcmAccessTimingParameter)
0x84	NRC "ServiceNotSupported" (Dsp_DcmSecuredDataTransmission)
0x85	Dsp_DcmControlDTCSetting
0x86	NRC "ServiceNotSupported" (Dsp_DcmResponseOnEvent)
0x87	NRC "ServiceNotSupported" (Dsp_DcmLinkControl)

0x22	Dsp_DcmReadDataByIdentifier
0x23	NRC "ServiceNotSupported" (Dsp_DcmReadMemoryByAddress)
0x24	Dsp_DcmReadScalingDataByIdentifier
0x2A	Dsp_DcmReadDataByPeriodicIdentifier
0x2C	Dsp_DcmDynamicallyDefineDataIdentifier
0x2E	Dsp_DcmWriteDataByIdentifier
0x3D	NRC "ServiceNotSupported" (Dsp_DcmWriteMemoryByAddress)
0x14	Dsp_DcmClearDiagnosticInformation
0x19	Dsp_DcmReadDTCInformation
0x2F	Dsp_DcmInputOutputControlByIdentifier
0x31	Dsp_DcmRoutineControl
0x34	NRC "ServiceNotSupported" (Dsp_DcmRequestDownload)
0x35	NRC "ServiceNotSupported" (Dsp_DcmRequestUpload)
0x36	NRC "ServiceNotSupported" (Dsp_DcmTransferData)
0x37	NRC "ServiceNotSupported" (Dsp_DcmRequestTransferExit)

Table 5: List of function naming examples for requests of ISO14229-1

ISO15031-5

Service Identifier	Function name examples for <DiagnosticService>
0x01	Dsp_DcmRequestCurrentDiagnosticData
0x02	Dsp_DcmRequestFreezeFrameData
0x03	Dsp_DcmRequestEmissionRelatedDTC
0x04	Dsp_DcmClearEmissionRelatedInformation
0x05	Dsp_DcmRequestOxySenMoniTestResults
0x06	Dsp_DcmRequestOnBoardMoniTestResults
0x07	Dsp_DcmReqEmissionRelDTCDrivCyc
0x08	Dsp_DcmRequestControl
0x09	Dsp_DcmRequestVehicleInformation

Table 6: List of function naming examples for requests of ISO15031-5

8.8 Proposed internal interfaces

8.8.1 DSL (Diagnostic Session Layer) Functions

Security Level

8.8.1.1 Dcm_SetSecurityLevel

Service name:	Dcm_SetSecurityLevel
Syntax:	<pre>void Dcm_SetSecurityLevel (Dcm_SecLevelType SecurityLevel)</pre>
Service ID:	0x0E
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant

Parameters (in):	SecurityLevel	New security level value Conversion formula to calculate SecurityLevel out of tester requested SecurityAccessType parameter: $SecurityLevel = (SecurityAccessType + 1) / 2$ Content is according to "securityAccessType" parameter of SecurityAccess request (see [11])
Parameters (out):	None	
Return value:	None	
Description:	to set the new security level value in DCM	
Caveats:	None	
Configuration:	None	

Session State Access

8.8.1.2 Dcm_SetSesCtrlType

Service name:	Dcm_SetSesCtrlType	
Syntax:	<pre>void Dcm_SetSesCtrlType (Dcm_SesCtrlType SesCtrlType)</pre>	
Service ID:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant	
Parameters (in):	SesCtrlType	New session control type value Content is according to "diagnosticSessionType" parameter of DiagnosticSessionControl request (see [11])
Parameters (out):	None	
Return value:	None	
Description:	to set the new session control type value in DCM	
Caveats:	None	
Configuration:	None	

Session Timing Access

8.8.1.3 Dcm_GetSesTimingValues

Service name:	Dcm_GetSesTimingValues	
Syntax:	<pre>void Dcm_GetSesTimingValues (Dcm_TimerModeType TimerMode, Dcm_TimerServerType* TimerServerCurrent)</pre>	
Service ID:	0x0A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TimerMode	Configures, which set of timing parameters needs to be read (see description of Parameters)
Parameters (in/out):	TimerServerCurrent	TimerServerCurrent is the address where the Set of timing parameters (see description of Parameters) have to be

	written
Return value:	None
Description:	This function is used to read dependent on the TimerMode, the current timing parameter value, default timing parameters or the timing parameter limits.
Caveats:	This API is only used for the service "AccessTimingParameters" which is not used within UDS on CAN
Configuration:	None

8.8.1.4 Dcm_PrepareSesTimingValues

Service name:	Dcm_PrepareSesTimingValues
Syntax:	<pre>Dcm_StatusType Dcm_PrepareSesTimingValues (const Dcm_TimerServerType* TimerServerNew)</pre>
Service ID:	0x0B
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	TimerServerNew Set of new timing parameters (see description of Parameters)
Parameters (out):	None
Return value:	Result of the preparation / examination: DCM_E_OK: preparation was successful DCM_E_TI_PREPARE_LIMITS: requested values are not within the defined limits DCM_E_TI_PREPARE_INCONSTENT: requested values are not consistent (e.g. P2min not smaller than P2max)
Description:	to prepare setting of new timing parameters (examination, if new values are within the limits/configured timings).
Caveats:	This API is only used for the service "AccessTimingParameters" which is not used within UDS on CAN
Configuration:	None

8.8.1.5 Dcm_SetSesTimingValues

Service name:	Dcm_SetSesTimingValues
Syntax:	<pre>void Dcm_SetSesTimingValues (const Dcm_TimerServerType* TimerServerNew)</pre>
Service ID:	0x0C
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	TimerServerNew Set of new timing parameters (see description of Parameters)
Parameters (out):	None
Return value:	None
Description:	Setting of the new timing parameters. This function can be requested by the application after successful preparation of the timing parameters. This request is allowed only in the response confirmation function (Dsp_DcmConfirmation), to guarantee that the response is send, before the new timing parameter are set.

Caveats:	This API is only used for the service "AccessTimingParameters" which is not used within UDS on CAN
Configuration:	None

ROE handling

8.8.1.6 Dcm_ResponseOnOneEvent

Service name:	Dcm_ResponseOnOneEvent	
Syntax:	<pre>Dcm_StatusType Dcm_ResponseOnOneEvent (Const Dcm_MsgType MsgPtr, Dcm_MsgLenType MsgLen, PduIdType DcmRxPduId)</pre>	
Service ID:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Not reentrant. This function may not be called twice at one time. One processing shall be finished before the next sending is ordered. DCM is responsible to handle any kind of synchronizing.	
Parameters (in):	MsgPtr	Content of requested service (e.g. SID ReadDataByIdentifier with DataIdentifier 0x1234 -> buffer content: 0x22, 0x12, 0x34)
	MsgLen	Length in bytes
	DcmRxPduId	Only if Type1 is used. Type1 requires that the ROE response (on event) is transmitted with the same Source- and TargetAddress as the physical response of the ROE request. Therefore the application has to provide the DcmRxPduId given in the pMsgContext of the ROE Service function.
Parameters (out):	None	
Return value:	Permission of DCM. DCM_E_OK: ResponseOnOneEvent request is accepted by DCM and the request data is copied into internal reception buffer, DCM_E_ROE_NOT_ACCEPTED: ResponseOnOneEvent request is not accepted by DCM (e.g. current pending request is not finished)	
Description:	This API provides not the processing of the ResponseOnEvent service. But it provides the processing of one event, requested internally in the DCM. A sequence diagram showing more details can be found in chapter 9.2.5 Process single event-triggered response of ResponseOnEvent	
Caveats:	None	
Configuration:	DcmDslROEEnabled	

Periodic transmission handling

8.8.1.7 Dcm_ResponseOnOneDataByPeriodicId

Service name:	Dcm_ResponseOnOneDataByPeriodicId
Syntax:	<pre>Dcm_StatusType Dcm_ResponseOnOneDataByPeriodicId (</pre>

	uint8 PeriodicId)
Service ID:	0x11
Sync/Async:	Synchronous
Reentrancy:	Not reentrant. Function may not be called reentrantly (recursively). Only one processing at one time. DCM is responsible to handle any kind of queuing.
Parameters (in):	PeriodicId Requested Periodic Data Identifier DataIdentifier is calculated in the following way: High Byte: 0xF2, Low Byte: PeriodicId
Parameters (out):	None
Return value:	Permission of DCM. DCM_E_OK: Periodic transmission request is accepted by DCM, DCM_E_PERIODICID_NOT_ACCEPTED: Periodic transmission request is not accepted by DCM (e.g. old Periodic transmission is not finished)
Description:	This API provides not the processing of the ReadDataByPeriodicID service. But it provides the processing of one periodic ID event, requested internally in the DCM. The frequency of calling this function depends on the rate given in the original ReadDataByPeriodicID request (parameter transmissionMode). A sequence diagram showing more details can be found in chapter 9.2.4 Process single response of ReadDataByPeriodicIdentifier
Caveats:	None
Configuration:	DcmDslPeriodicTransEnabled

8.8.2 Dsp_DcmConfirmation

Service name:	Dsp_DcmConfirmation
Syntax:	void Dsp_DcmConfirmation (Dcm_IdContextType idContext, PduIdType dcmRxPduId, Dcm_ConfirmationStatusType status)
Service ID:	0x15
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	idContext Current context identifier which can be used to retrieve the relation between request and confirmation. Within the confirmation, the Dcm_MsgContext is no more available, so the idContext can be used to represent this relation. The idContext is also part of the Dcm_MsgContext. dcmRxPduId DcmRxPduId on which the request was received. The source of the request can have consequences for message processing. status Status indication about confirmation (differentiate failure indication and normal confirmation) / The parameter "Result" of "Dcm_TxConfirmation" shall be forwarded to status depending if a positive or negative response was sent.
Parameters (out):	None

Return value:	None
Description:	<p>This function confirms the successful transmission or a transmission error of a diagnostic service.</p> <p>This is the right time to perform any application state transitions.</p> <p>The idContext and the dcmRxDuldare required to identify the message which was processed. This information can be used to figure out the relation to any data provided by the application.</p> <p>The DSP receives this confirmation.</p>
Caveats:	<p>This function shall be called outside the interrupt context. This means that there can be some latency between successful transmission (or transmission error) and signaling to the application.</p> <p>Please note that the Dcm_MsgContext data structure and the buffers are no more valid when this function is called (because the message is already sent and the resources are already cleaned up here).</p>
Configuration:	None

8.8.3 Dcm_SetNegResponse

Service name:	Dcm_SetNegResponse	
Syntax:	<pre>void Dcm_SetNegResponse (Const Dcm_MsgContextType* pMsgContext, Dcm_NegativeResponseCodeType ErrorCode)</pre>	
Service ID:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	pMsgContext	Message-related information for one diagnostic protocol identifier.
	ErrorCode	Error code to be set. This error code will be taken as negative response code (NRC) (see description).
Parameters (out):	None	
Return value:	None	
Description:	<p>If this function is called, a negative response will be sent instead of a positive response. The negative response code is taken from the first call – duplicate calls are allowed but the errorCode transmitted is ignored.</p> <p>The transmission of the negative response will be performed after calling Dcm_ProcessingDone.</p> <p>For a list of negative response codes supported, please refer chapter 8.3.1.9 Dcm_NegativeResponseCode</p>	
Caveats:		
Configuration:	None	

8.8.3.1 Dcm_ProcessingDone

Service name:	Dcm_ProcessingDone	
Syntax:	<pre>void Dcm_ProcessingDone (Const Dcm_MsgContextType* pMsgContext</pre>	

)
Service ID:	0x14
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	pMsgContext Message-related information for one diagnostic protocol identifier.
Parameters (out):	None
Return value:	None.
Description:	<p>When this function is called, a response will be sent based on the data contained in pMsgContext. If Dcm_SetNegResponse was called before, a negative response is sent, otherwise a positive response will be sent. The DSP does not have to care about any timing requirement to process a request.</p> <p>For diagnostic experts: Between the arrival of a request (Dsp_Dcm<DiagnosticService>) and finishing the corresponding response (Dcm_Processing_Done), busy-acknowledges (negative response with response code 0x78) are sent by the DSL automatically.</p>
Caveats:	
Configuration:	None

8.8.3.1.1 Dsp_Dcm<DiagnosticService>

Service name:	Dsp_Dcm<DiagnosticService>
Syntax:	<pre>void Dsp_Dcm<DiagnosticService> (Dcm_MsgContextType* pMsgContext)</pre>
Service ID:	0x12
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant (this function will be only called once at a time)
Parameters (in):	pMsgContext Message-related information for one diagnostic protocol identifier.
Parameters (out):	None
Return value:	None
Description:	<p>Callback function. DCM shall call this callback function as soon as valid message is received on relevant DcmRxDuld.</p> <p>The usecase of multiple diagnostic protocols will be possible by using different arguments and the function shall be programmed in a way it is reentrant. This feature is not supported for current version. Caller (lower layer) is responsible for the lifetime of the argument pMsgContext. The Dsp does not have to care about any timing requirement to process a request</p>
Caveats:	None
Configuration:	Only available if diagnostic corresponding protocol service is supported. DcmDsdSidFunction

8.8.3.2 Dcm_StartPagedProcessing

Service name:	Dcm_StartPagedProcessing
Syntax:	void Dcm_StartPagedProcessing (Const Dcm_MsgContextType* pMsgContext)
Service ID:	0x18
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	pMsgContext Message-related information for one diagnostic protocol identifier.
Parameters (out):	None
Return value:	None
Description:	With this API, the DSP gives the complete response length to DCM and starts PagedBuffer handling. Complete response length information (in bytes) is given in pMsgContext->resDataLen Callback functions are used to provide paged buffer handling in DSP. More information can be found in the sequence chart in chapter 9.3.5 Process Service Request with PagedBuffer.
Caveats:	This information is needed e.g. at UDSONCAN for sending FirstFrame This API starts no transmission! This API is only needed at PagedBuffer handling
Configuration:	DcmPagedBufferEnabled

8.8.3.3 Dcm_ProcessPage

Service name:	Dcm_ProcessPage
Syntax:	void Dcm_ProcessPage (Dcm_MsgLenType FilledPageLen)
Service ID:	0x1A
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	FilledPageLen By application filled size (in Bytes) of Page Note: max of FilledPageLen =DCM_BUFFER_SIZE – 1 for the first response message
Parameters (out):	None
Return value:	None
Description:	DSP requests transmission of filled page More information can be found in the sequence chart in chapter 9.3.5 Process Service Request with PagedBuffer.
Caveats:	This API is only needed at PagedBuffer handling
Configuration:	DcmPagedBufferEnabled

8.8.3.3.1 Dsp_DcmUpdatePage

Service name:	Dsp_DcmUpdatePage
Syntax:	void Dsp_DcmUpdatePage (Dcm_MsgType PageBufPtr, Dcm_MsgLenType PageLen

)
Service ID:	0x19
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	PageBufPtr Pointer to Page buffer
	PageLen available Page buffer size (in Bytes)
Parameters (out):	None
Return value:	None
Description:	With this API, DSP is requested to fill next page. Page buffersize and Page buffer is given to DSP More information can be found in the sequence chart in chapter 9.3.5 Process Service Request with PagedBuffer.
Caveats:	This API is only needed at PagedBuffer handling
Configuration:	DcmPagedBufferEnabled

8.8.3.3.2 Dsp_DcmCancelPagedBufferProcessing

Service name:	Dsp_DcmCancelPagedBufferProcessing
Syntax:	void Dsp_DcmCancelPagedBufferProcessing(void)
Service ID:	0x1B
Sync/Async:	Synchronous
Reentrancy:	Non-Reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	DCM informs DSP, that processing of paged buffer was cancelled due to errors DSP is not allowed to process further on pagedBuffer handling, when CancelPagedBufferProcessing is indicated.
Caveats:	This API is only needed at PagedBuffer handling
Configuration:	DcmPagedBufferEnabled

8.8.4 Dcm_DcmDiagnosticSessionControl

This service is used to enable different diagnostic sessions in the server. For details, see chapter 7.2.3.6.1 Diagnostic Session Control.

8.8.5 Dcm_DcmTesterPresent

This service is used to keep one or multiple servers in a diagnostic session being different than the defaultSession. For details, see chapter 7.2.3.6.2 Tester Present.

8.8.6 Dcm_DcmSecurityAccess

The purpose of this service is to provide a means to access data and/or diagnostic services, which have restricted access for security, emissions, or safety reasons. For details, see chapter 7.2.3.6.3 SecurityAccess.

8.9 DSP (Diagnostic Service Processing)

8.9.1 Interface DSP – DEM

Dcm002: The DEM capsulates all functionalities relating to error detection and storage. An external test tool may require all kinds of error related information by using the error memory access diagnostic services defined by ISO 15031-5 and ISO 14229-1. For this reason the DEM provides a well-defined interface for accessing all error related information. As described in section 7.2.3.5 the DSP uses the DEM interface to assemble all the responses on external tester requests relating to error memory information.

For a detailed interface description please refer to the appropriate section in AUTOSAR_SWS_DEM [7] chapter 8.4.2 - Interface DCM ↔ DEM.

8.10 AUTOSAR Services implemented by DCM

8.10.1 Scope of this chapter

This chapter is an addition to the specification of the DCM module. Whereas the other parts of the specification define the behavior and the C-interfaces of the corresponding basic software module, this chapter formally specifies the corresponding AUTOSAR Service in terms of the SWC Template. The interfaces described here will be visible on the VFB and are used to generate the RTE between application software and the DCM module.

8.10.2 Specification of the Ports and Port Interfaces for Diagnostic Services

This chapter specifies the ports and port interfaces which are needed in order to operate the DCM functionality over the VFB. Note that there are ports on both sides of the RTE: The SW-C description of the DCM will define the ports below the RTE. Each SW-Component, which uses the Service, must contain “service ports” in its own SW-C description which will be connected to the ports of the DCM, so that the RTE can be generated.

8.10.2.1 General Approach

8.10.2.2 Data Types

This chapter describes the data types which will be used in the port interfaces for the Diagnostic Communication Manager services requests and session handling.

The data types `UInt8`, `UInt16` and `Boolean` used in the interfaces refers to the basic AUTOSAR data types specified in [SWCTypes].

The data type `ProtocolType`, which indicates information on diagnostic protocol potentially used, which will be defined by the SW-C description in the following way:

```
IntegerType ProtocolType {
    LOWER-LIMIT = 0
    UPPER-LIMIT = 255
};
0 -> DCM_OBD_ON_CAN
1 -> DCM_UDS_ON_CAN
2 -> DCM_UDS_ON_FLEXRAY
3 -> DCM_ROE_ON_CAN
4 -> DCM_ROE_ON_FLEXRAY
5 -> DCM_PERIODICTRANS_ON_CAN
6 -> DCM_PERIODICTRANS_ON_FLEXRAY
7 -> Reserved for further AUTOSAR implementation (up to
239)
240 -> Reserved for SW supplier specific (up to 255)
```

The data type `Dcm_NegativeResponseCodeType`, which indicates the cause of a negative result to a diagnostic request, which will be defined by the SW-C description in the following way:

```
IntegerType Dcm_NegativeResponseCodeType{
    LOWER-LIMIT = 0
    UPPER-LIMIT = 255
};
```

Via an associated `CompuMethod` the following constants are defined within this type:

```
0x10 -> DCM_E_GENERALREJECT
0x11 -> DCM_E_SERVICENOTSUPPORTED
0x12 -> DCM_E_SUBFUNCTIONNOTSUPPORTED
0x13 -> DCM_E_INCORRECTMESSAGELENGTHORINVALIDFORMAT
0x14 -> DCM_E_RESPONSE_TOO_LONG
0x21 -> DCM_E_BUSYREPEATREQUEST
0x22 -> DCM_E_CONDITIONSNOTCORRECT
0x24 -> DCM_E_REQUESTSEQUENCEERROR
0x31 -> DCM_E_REQUESTOUTOFRANGE
```

0x33 -> DCM_E_SECURITYACCESSDENIED
0x35 -> DCM_E_INVALIDKEY
0x36 -> DCM_E_EXCEEDNUMBEROFATTEMPTS
0x37 -> DCM_E_REQUIREDTIMEDELAYNOTEXPIRED
0x70 -> DCM_E_UPLOADDOWNLOADNOTACCEPTED
0x71 -> DCM_E_TRANSFERDATASUSPENDED
0x72 -> DCM_E_GENERALPROGRAMMINGFAILURE
0x73 -> DCM_E_WRONGBLOCKSEQUENCECOUNTER
0x78 -> DCM_E_REQUESTCORRECTLYRECEIVED-RESPONSEPENDING
0x7E -> DCM_E_SUBFUNCTIONNOTSUPPORTEDINACTIVESSESSION
0x7F -> DCM_E_SERVICENOTSUPPORTEDINACTIVESSESSION
0x81 -> DCM_E_RPMTOOHIGH
0x82 -> DCM_E_RPMTOOLOW
0x83 -> DCM_E_ENGINEISRUNNING
0x84 -> DCM_E_ENGINEISNOTRUNNING
0x85 -> DCM_E_ENGINERUNTIME TOOLOW
0x86 -> DCM_E_TEMPERATURETOOHIGH
0x87 -> DCM_E_TEMPERATURETOOLOW
0x88 -> DCM_E_VEHICLESPEEDTOOHIGH
0x89 -> DCM_E_VEHICLESPEEDTOOLOW
0x8A -> DCM_E_THROTTLE_PEDALTOOHIGH
0x8B -> DCM_E_THROTTLE_PEDALTOOLOW
0x8C -> DCM_E_TRANSMISSIONRANGENOTINNEUTRAL
0x8D -> DCM_E_TRANSMISSIONRANGENOTINGEAR
0x8F -> DCM_E_BRAKESWITCH_NOTCLOSED
0x90 -> DCM_E_SHIFTERLEVERNOTINPARK
0x91 -> DCM_E_TORQUECONVERTERCLUTCHLOCKED
0x92 -> DCM_E_VOLTAGETOOHIGH
0x93 -> DCM_E_VOLTAGETOLOW

The data type `SesCtrlType`, which defines the session type, which will be defined by the SW-C description in the following way:

```
IntegerType SesCtrlType {
    LOWER-LIMIT = 1
    UPPER-LIMIT = 255
};
1 -> DEFAULT_SESSION
2 -> PROGRAMMING_SESSION
3 -> EXTENDED_DIAGNOSTIC_SESSION
4 -> SAFETY_SYSTEM_DIAGNOSTIC_SESSION
5 -> Configuration dependent (up to 127)
128 -> Reserved by document (up to 254)
255 -> ALL_SESSION_LEVEL
```

The data type `SecLevelType`, which defines the security level type, which will be defined by the SW-C description in the following way:

```
IntegerType SecLevelType {
    LOWER-LIMIT = 0
    UPPER-LIMIT = 255
};
0 -> DCM_SEC_LEV_LOCKED
1 -> DCM_SEC_LEV_L1
3 -> Configuration dependent (up to 127)
128 -> Reserved by document (up to 254)
255 -> DCM_SEC_LEV_ALL
```

8.10.2.3 Port Interface of Diagnostic Services

The operations correspond to the function calls of the DCM C-API, the pseudo code occurs some refinement to be translated in XML. The prefix “DCM_” doesn’t appear anymore due to the fact that part of the interface description.

The following interface provides operation to access data identifiers, in order to fulfil process of the diagnostic services: `ReadDataByIdentifier`, `WriteDataByIdentifier`, `ReadDataByPeriodicIdentifier`, `InputOutputControlByIdentifier`, `GetScalingInformation`.

One `DidServices` interface will be generated for each DID.

```
ClientServerInterface DidServices_<DID> {

PossibleErrors {
    E_OK = 0,
    E_NOT_OK = 1,
    E_PENDING = 10,
};

// the next operation is provided if the DID can be read
ReadData (OUT <DID specific type> data, ERR{DCM_E_PENDING})
```

```

// the next operation is provided if the DID can be written
WriteData (IN <DID specific type> data, IN uint16 dataLength,
OUT NegativeResponseType ErrorCode, ERR{DCM_E_PENDING})

// the next operation is provided if the DID has a variable
data length
ReadDataLength (OUT uint16 DidLength, ERR{DCM_E_PENDING})
// the next operation is provided if the DID can be read
ConditionCheckRead (OUT NegativeResponseType ErrorCode,
ERR{DCM_E_PENDING})

// the next operation is provided if the DID can be written
ConditionCheckWrite (OUT NegativeResponseType ErrorCode,
ERR{DCM_E_PENDING})

// the next operation is provided optionally
ReturnControlToECU (<Control specific parameters>,
ERR{DCM_E_PENDING})

// the next operation is provided optionally
ResetToDefault (<Control specific parameters>,
ERR{DCM_E_PENDING})

// the next operation is provided optionally
FreezeCurrentState (<Control specific parameters>,
ERR{DCM_E_PENDING})

// the next operation is provided optionally
ShortTermAdjustment (<Control specific parameters>,
ERR{DCM_E_PENDING})

// the next operation is provided optionally
GetScalingInformation (OUT uint8 ScalingInfo[<Scaling Info
size>], ERR{DCM_E_PENDING} )

};

```

The following interface provides operation to control diagnostic routine, in order to fulfil process of the diagnostic services: RoutineControl
One RoutineServices interface will be generated for each Routine.

```

ClientServerInterface RoutineServices_<RoutineName> {

PossibleErrors {
    E_OK = 0,
    E_NOT_OK = 1,
    E_PENDING = 10,
};

// the next operation is always provided

```

```

Start      (<Routine      specific      parameters>,      OUT
NegativeResponseCodeType ErrorCode, ERR{DCM_E_PENDING})

// the next operation is provided if the routine can be
stopped
Stop      (<Routine      specific      parameters>,      OUT
NegativeResponseCodeType ErrorCode, ERR{DCM_E_PENDING})

// the next operation is provided if the status of the
routine can be requested
RequestResults (<Routine      specific      parameters>,      OUT
NegativeResponseCodeType ErrorCode, ERR{DCM_E_PENDING})
};

```

The following interface provides operation to get information from DCM

```

ClientServerInterface DCMServices {
    PossibleErrors {
        E_OK = 0,
        E_NOT_OK = 1,
        E_PENDING = 10,
    };

    GetActiveProtocol      (OUT      ProtocolType      ProtocolId,
ERR{DCM_E_NOT_OK});
    GetSesCtrlType (OUT SesCtrlType SessionCtrlType, ERR{DCM_E_NOT_OK});
    GetSecurityLevel (OUT SecLevelType SecLevel, ERR{DCM_E_NOT_OK});
}

```

The following interface provides information on status of the protocol communication.

```

SenderReceiverInterface CallbackDCMRequestServices {
    PossibleErrors {
        E_OK = 0,
        E_NOT_OK = 1,
        E_PROTOCOL_NOT_ALLOWED = 5,
        E_PENDING = 10,
    };

    StartProtocol      (IN      ProtocolType      ProtocolID,ERR{
DCM_E_PENDING ,DCM_E_PROTOCOL_NOT_ALLOWED});
    StopProtocol      (IN      ProtocolType      ProtocolID,      ERR{
DCM_E_NOT_OK});
};

```

The following interface provides operation to get information from a SW-C

```

ClientServerInterface SecurityAccess_<Level> {
PossibleErrors {
    E_OK = 0,
    E_NOT_OK = 1,
    E_PENDING = 10,
    E_COMPARE_KEY_FAILED = 11,
};

    GetSeed (IN uint8 securityAccessDataRecord[<size>],OUT UInt8
Seed[<seed size>], OUT NegativeResponseCodeType ErrorCode
ERR{E_NOT_OK, DCM_E_PENDING});
    CompareKey (IN uint8 Key[<key size>], ERR{E_NOT_OK,
DCM_E_PENDING, DCM_E_COMPARE_KEY_FAILED});
};

```

```

ClientServerInterface SessionControl {
PossibleErrors {
    E_OK = 0,
    E_NOT_OK = 1,
    E_SESSION_NOT_ALLOWED = 4,
    E_PENDING = 10,
    E_FORCE_RCRRP = 12,
};

    GetSesChgPermission (IN SesCtrlType SesCtrlTypeActive, IN
SesCtrlType SesCtrlTypeNew, ERR{ DCM_E_NOT_OK, DCM_E_PENDING,
DCM_E_SESSION_NOT_ALLOWED,DCM_E_FORCE_RCRRP });
    ChangeIndication (IN SesCtrlType SesCtrlTypeOld, IN
SesCtrlType SesCtrlTypeNew, ERR{ DCM_E_NOT_OK});
    ConfirmationRespPend (IN Dcm_ConfirmationStatusType status, ERR{
DCM_E_NOT_OK});
};

```

```

ClientServerInterface RequestIndication {
PossibleErrors {
    E_OK = 0,
    E_NOT_OK = 1,
    E_REQUEST_NOT_ACCEPTED = 8,
    E_REQUEST_ENV_NOK = 9,
};

    Indication (IN UInt8 SID, IN UInt8 RequestData[<Data size>],
IN UInt16 DataSize, ERR{E_NOT_OK, DCM_E_PENDING,
DCM_E_REQUEST_NOT_ACCEPTED, DCM_E_REQUEST_ENV_NOK});
};

```

```

ClientServerInterface ResetService {
PossibleErrors {
    E_OK = 0,

```

```
E_NOT_OK = 1,  
E_PENDING = 10  
};  
  
EcuReset(IN uint8 resetType, ERR{E_NOT_OK, DCM_E_PENDING});  
};
```

8.10.3 Definition of the Service DCM

```
ServiceComponent Dcm {  
  
  RequirePort SecurityAccess_<Level> SecurityAccess_<SWC>;  
  RequirePort SecurityAccess_<Level> SecurityAccess_<SWC>;  
  ...  
  RequirePort SecurityAccess_<Level> SecurityAccess_<SWC>;  
  
  RequirePort SessionControl SessionControl _<SWC>;  
  RequirePort SessionControl SessionControl _<SWC>;  
  ...  
  
  RequirePort DidServices DidServices_<DID>;  
  RequirePort DidServices DidServices_<DID>;  
  ...  
  RequirePort RoutineServices RoutineServices_<RoutineName>;  
  RequirePort RoutineServices RoutineServices_<RoutineName>;  
  ...  
  ProvidePort DCMServices DCMServices_<SWC>;  
  ProvidePort DCMServices DCMServices_<SWC>;  
  ...  
  RequirePort CallbackDCMRequestServices CallbackDCMRequestServices_<SWC>;  
  RequirePort CallbackDCMRequestServices CallbackDCMRequestServices_<SWC>;  
  ...  
  RequirePort RequestIndication RequestIndication_<SWC>;  
  RequirePort RequestIndication RequestIndication_<SWC>;  
  ...  
  RequirePort ResetService ResetService_<SWC>;  
  RequirePort ResetService ResetService_<SWC>;  
  ...  
}
```


9 Sequence diagrams

9.1 Overview

For an overview please refer to chapter 7.1.1 DCM Architecture overview

Notes:

For clarification, the following sequence diagrams don't represent the full communication mechanism between DCM and Pdu-Router. This is to hold the sequence diagrams simple.

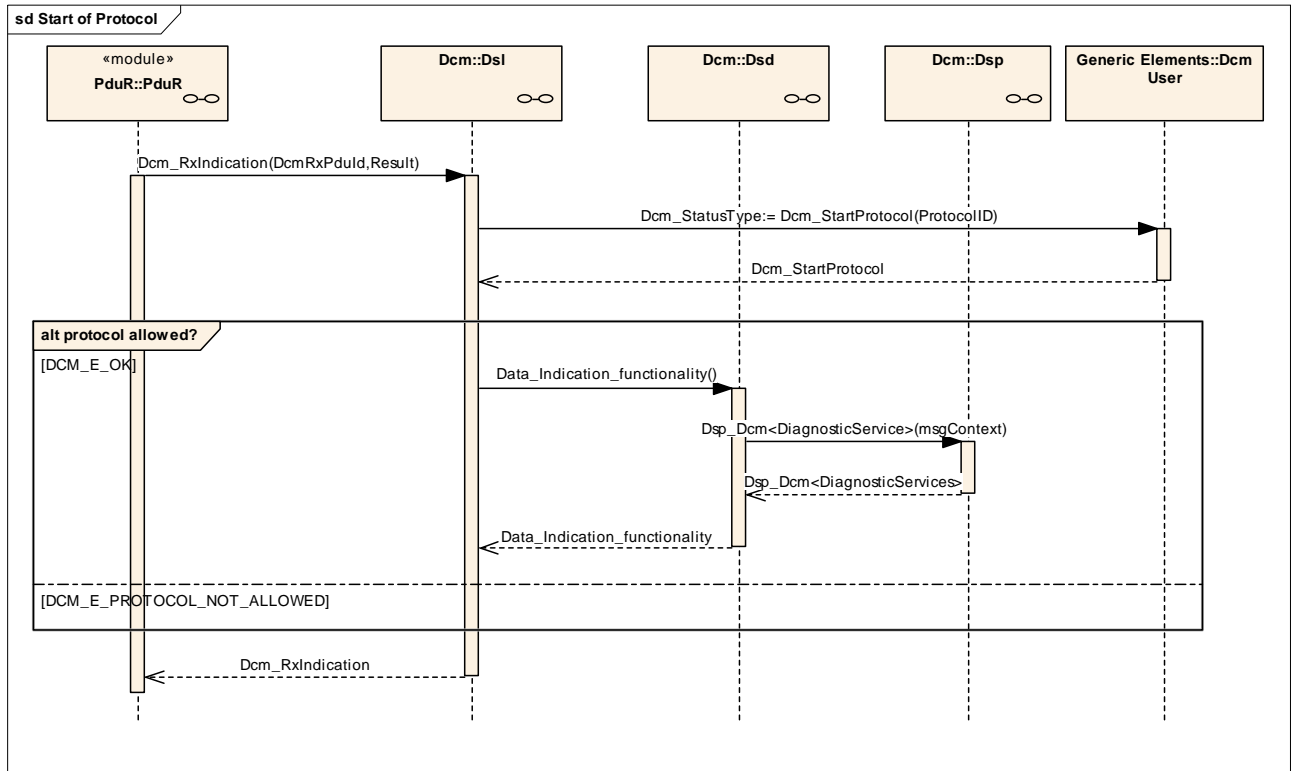
Before **Dcm_RxIndication()** call, Pdu-Router shall ask a buffer to DCM by using the **Dcm_ProvideRxBuffer()** service. This exchange is not shown on the next sequence diagrams.

As well, after a **PduR_DcmTransmit()** request from DCM to Pdu-Router, data exchanges with **Dcm_ProvideTxBuffer()** service, are not shown on the sequence diagrams.

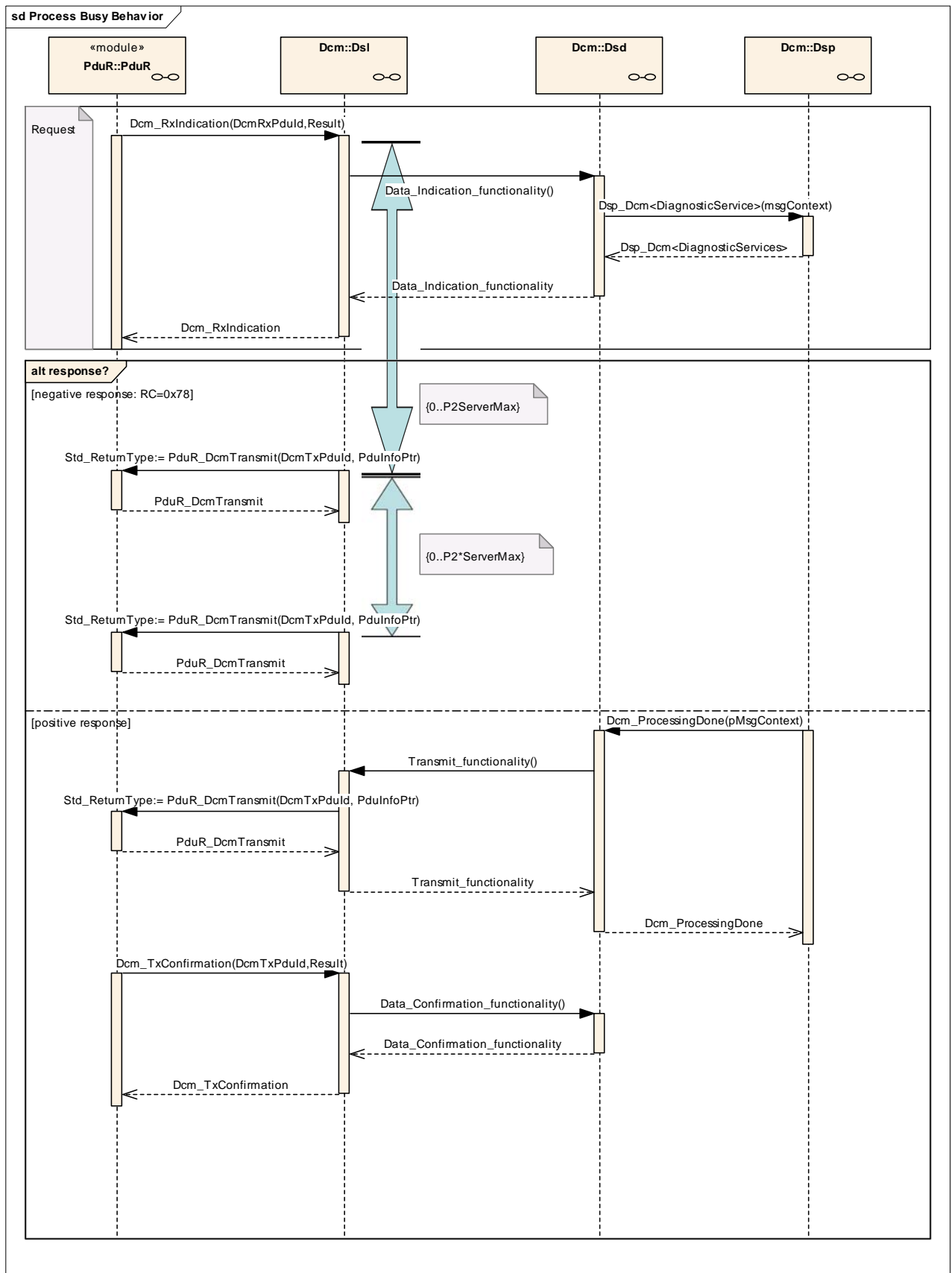
The Central Diagnostic Software component function **Dcm_StartProtocol()** shall be called with the very first diagnostic request.

9.2 DSL (Diagnostic Session Layer)

9.2.1 Start Protocol

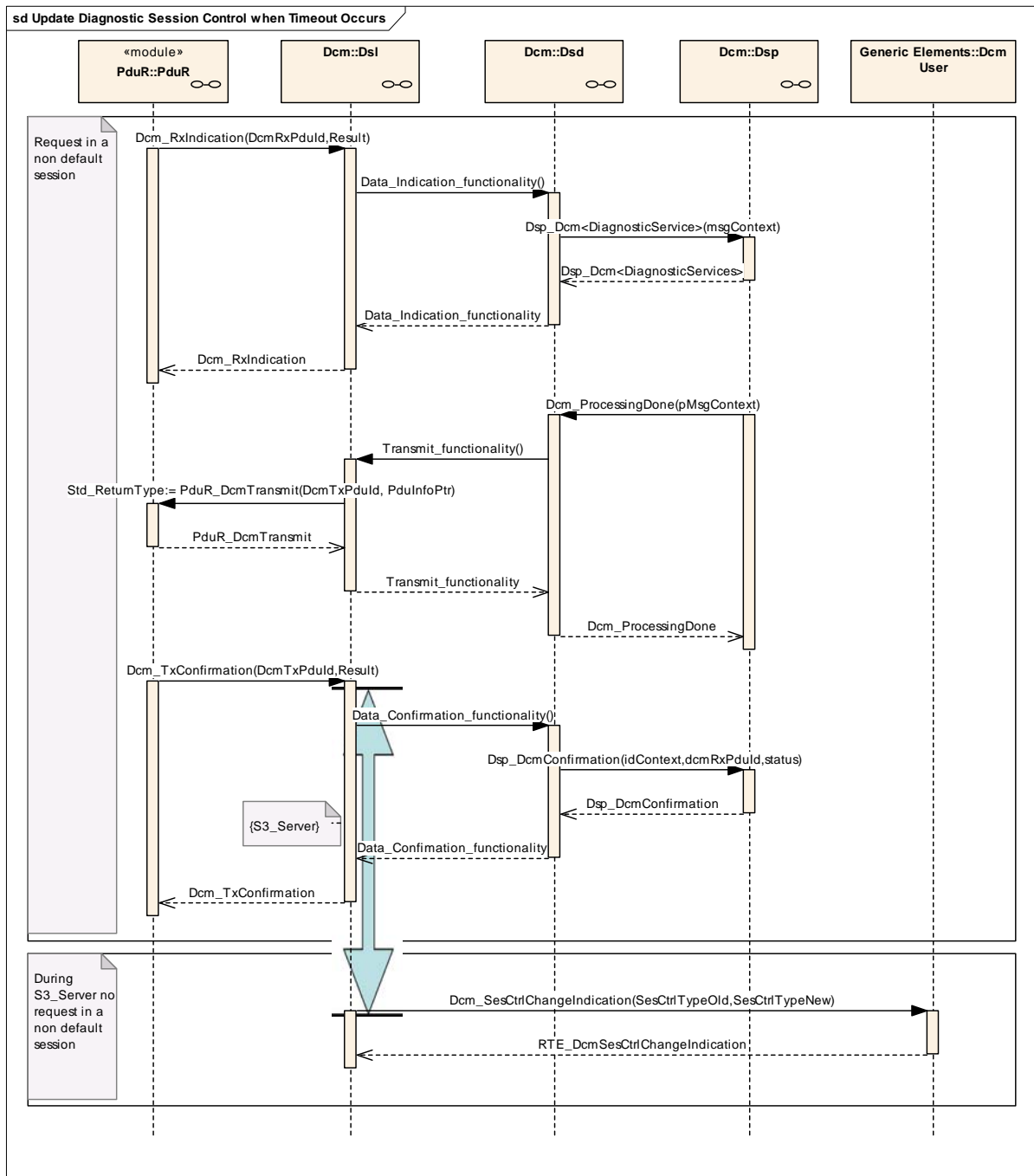


9.2.2 Process Busy behavior



Internally DSL calculates the time to response the tester. In case if the DSP don't close the request with **Dcm_ProcessingDone()** (in case of normal response handling) or **Dcm_ProcessPage()** (in case of PagedBuffer handling) during the P2ServerMax and / or P2*ServerMax the DSL sends negative response (requestCorectlyReceived-ResponsePending) independently.

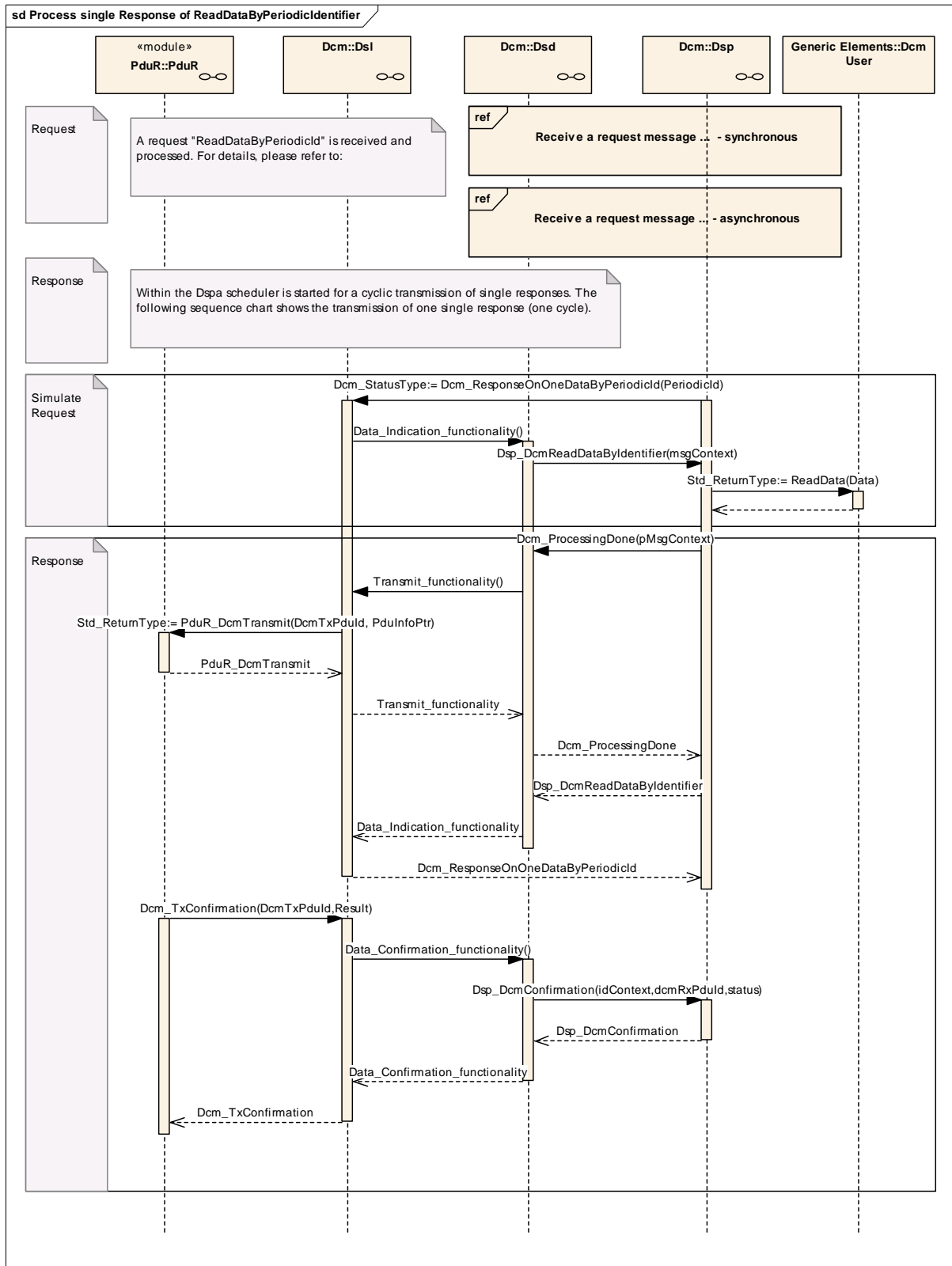
9.2.3 Update Diagnostic Session Control when timeout occurs



DSL resets session control value to default, if in a non-default session S3server timeout occurs.

S3server timeout timer will be started with every data confirmation from the PDU Router.

9.2.4 Process single response of ReadDataByPeriodicIdentifier



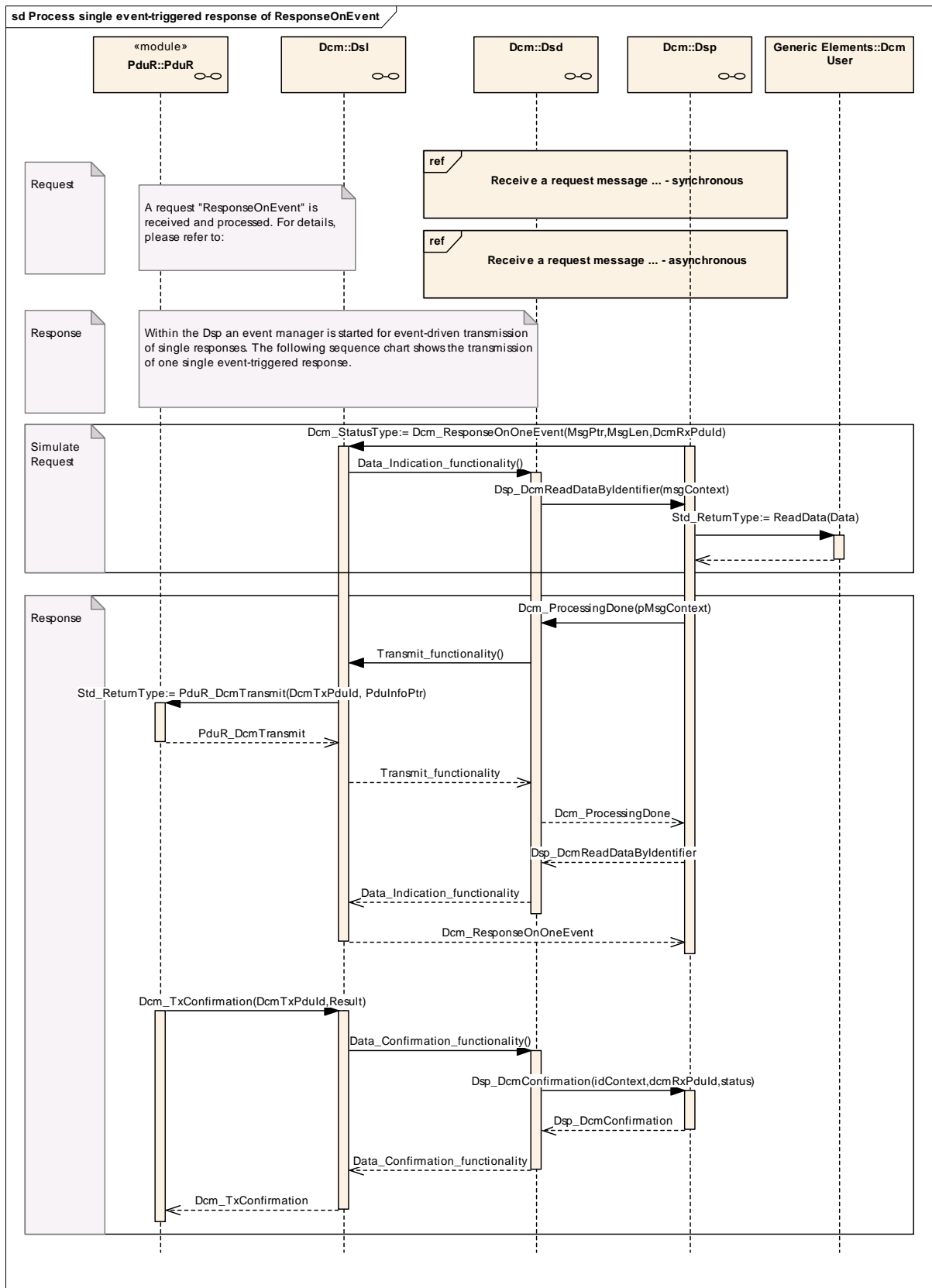
DSP requests sampling and transmission of Periodic Identifier data, when an event to Periodic Identifier occurs (i. e. a given time period is over). The DSP initiates the sending of one periodic identifier calling the function **Dcm_ResponseOneDataByPeriodicId()** provided by the DSL.

Within this function the DSL simulates a “ReadDataByIdentifier” request for the given PeriodicId. The High byte of the DataIdentifier shall be set to 0xF2 as specified in [11]) and the low byte is set to value of the PeriodicId.

The ReadData interface of the corresponding DID is called to get the DID value.

DCM is not able to receive another Periodic Identifier event request from DSP, unless the last periodic identifier event request is finished and the confirmation is received.

9.2.5 Process single event-triggered response of ResponseOnEvent



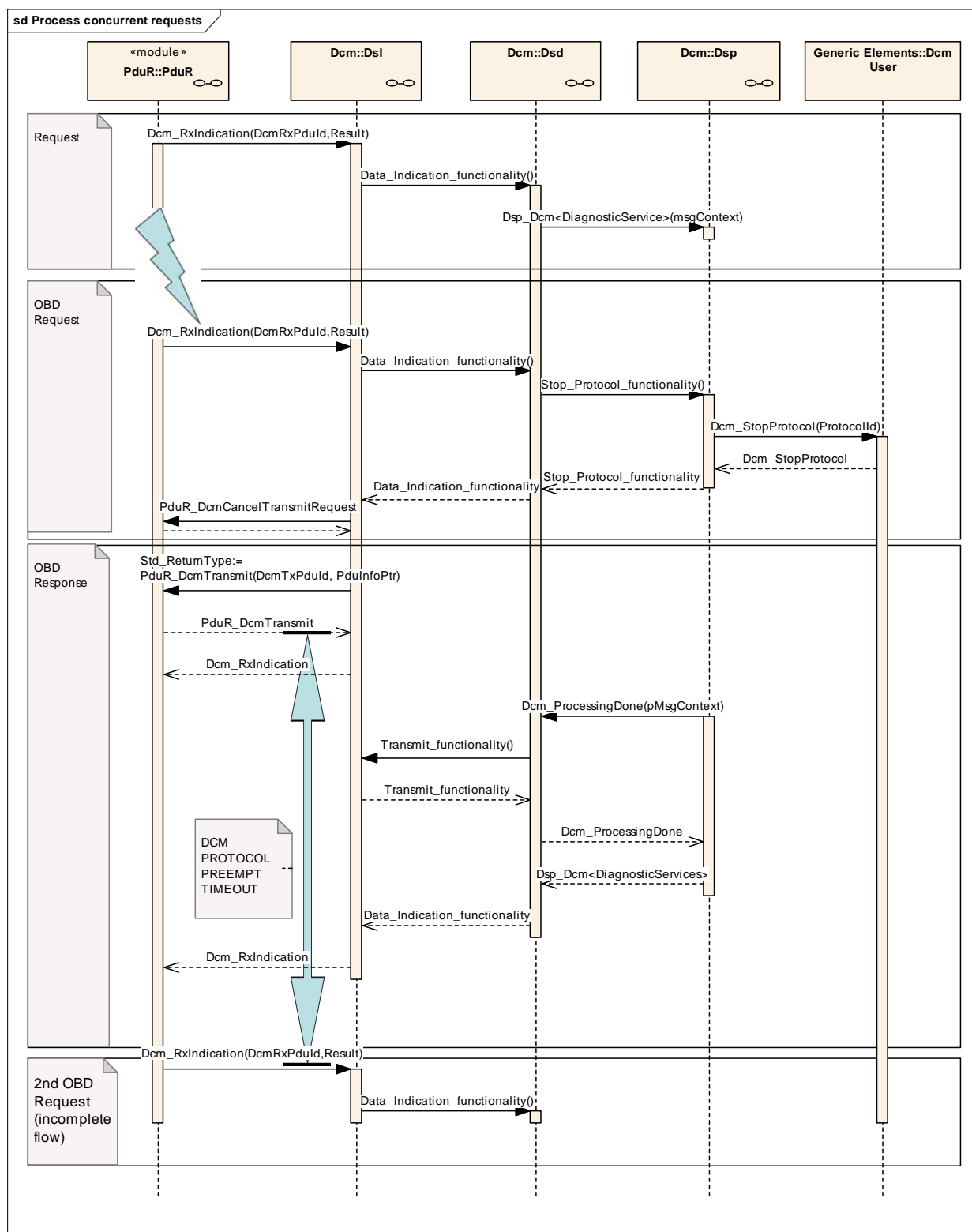
The DSP requests transmission of ResponseOnEvent data, when ROE event occurs (**Dcm_ResponseOnOneEvent()**).

Within this function the DSL simulates a corresponding diagnostic request, which have been registered before. This request will be processed by the DSP.

The ReadData interface of the corresponding DID is called to get the DID value.

The DCM will not be able to process multiple event-triggered responses at one time.

9.2.6 Process concurrent requests



On reception of OBD request in parallel to processing of a normal diagnostic request (e.g enhanced diagnostic protocol, customer diagnostic protocol), running diagnostic request will be preempted. This is due to the configured higher priority of OBD protocol (see configuration parameter **DcmDslProtocolPriority**).

The following is processed on reception of 1st OBD request:

- Application is informed of the protocol stop (done with **Dcm_StopProtocol()**) and to reset to a stable state (e.g. switch of digital IOs,..).
- Lower Layer is requested to cancel ongoing transmission on the same N-PDU (done with **PduR_DcmCancelTransmitRequest()**)
- DSL responses with a negative response "BusyRepeatRequest" (NRC 0x21) to OBD tester.
- Timeout tracking of application finishes is started (timeout value configured in parameter **DcmDslProtocolPreemptTimeout** of the preempting protocol (here OBD protocol))

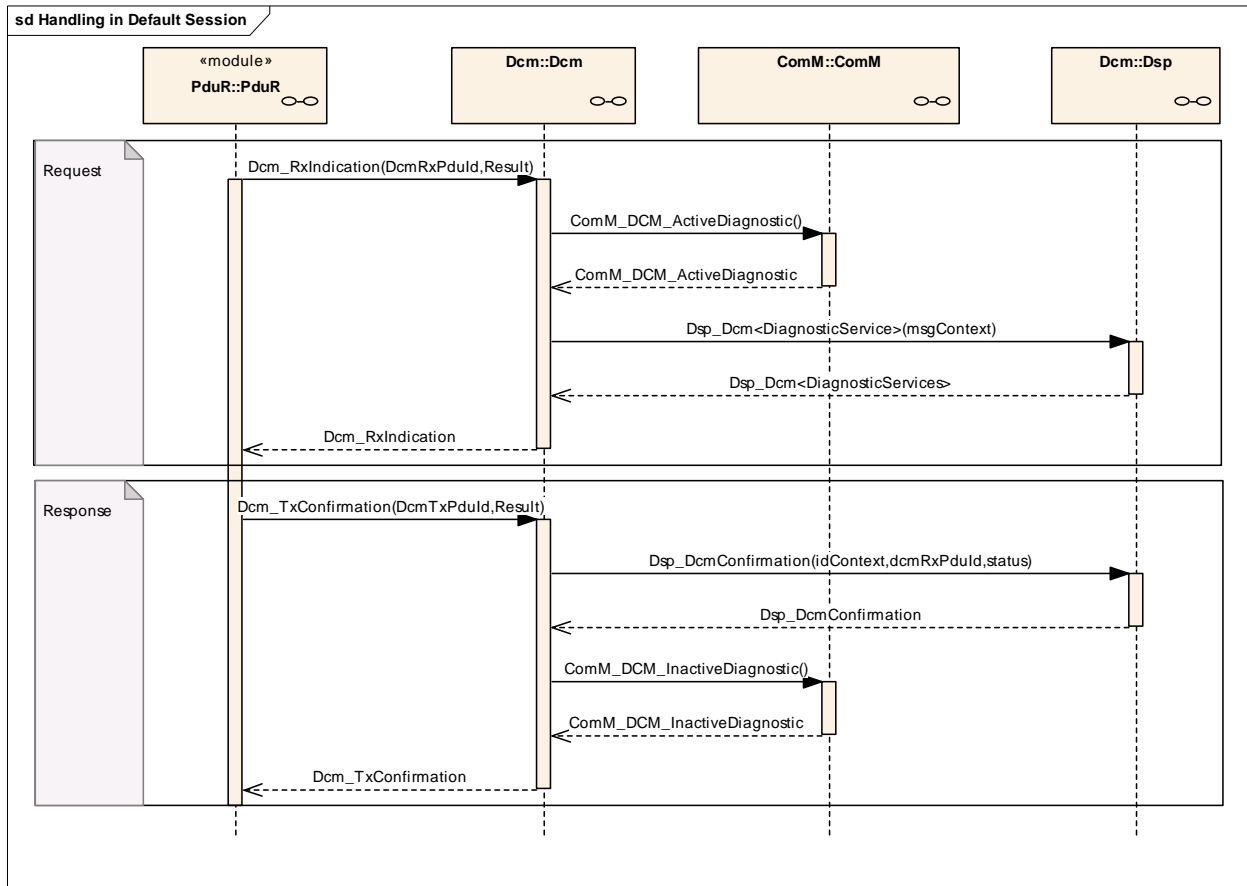
As long DSP is not finished (finish is indicated with **Dcm_ProcessingDone()**) or no timeout occurs, DSL responses with negative response "BusyRepeatRequest".

With receiving **Dcm_ProcessingDone()**, DSL will not transmit a response to old request. There will also not given any negative response to inform first tester about preemption of diagnostic request.

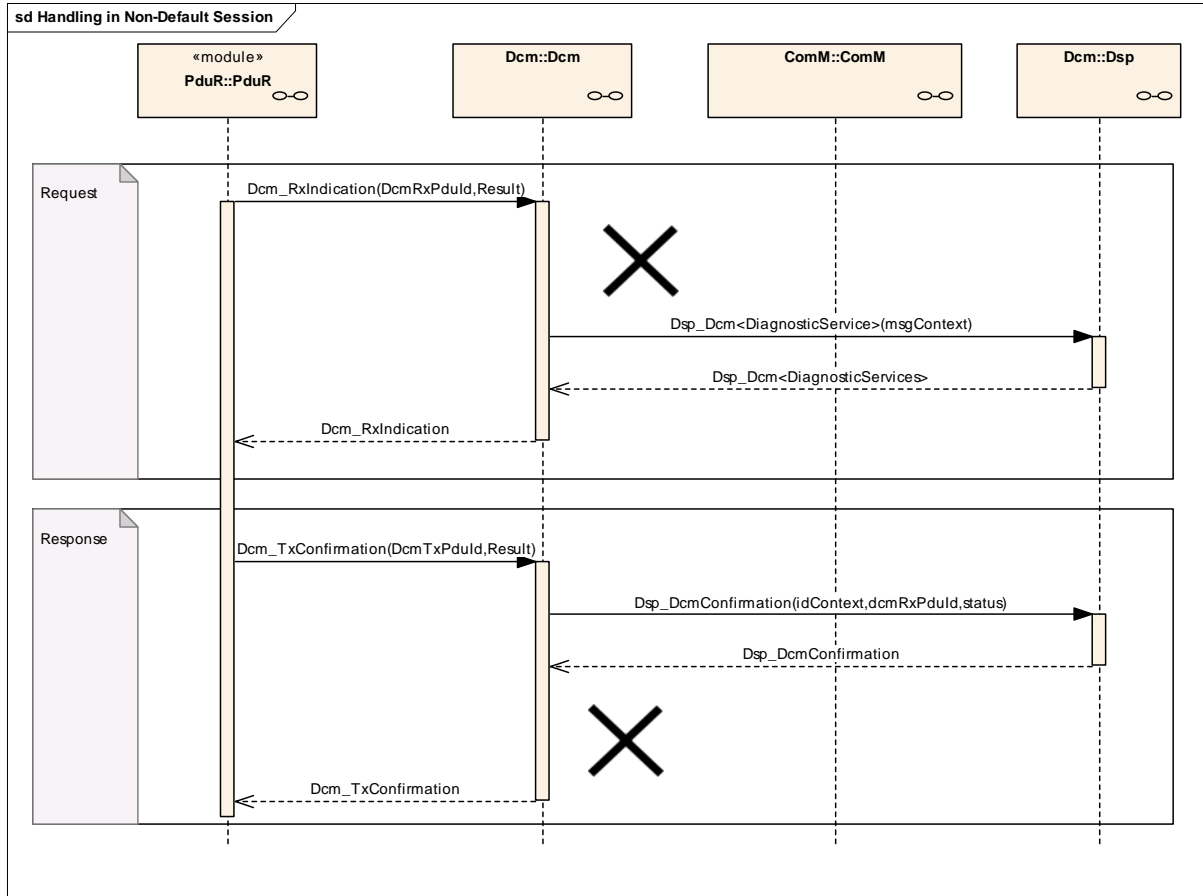
If DSP triggers no **Dcm_ProcessingDone()**, DSL runs into timeout and switches directly to further processing of preempting protocol.

9.2.7 Interface to ComManager

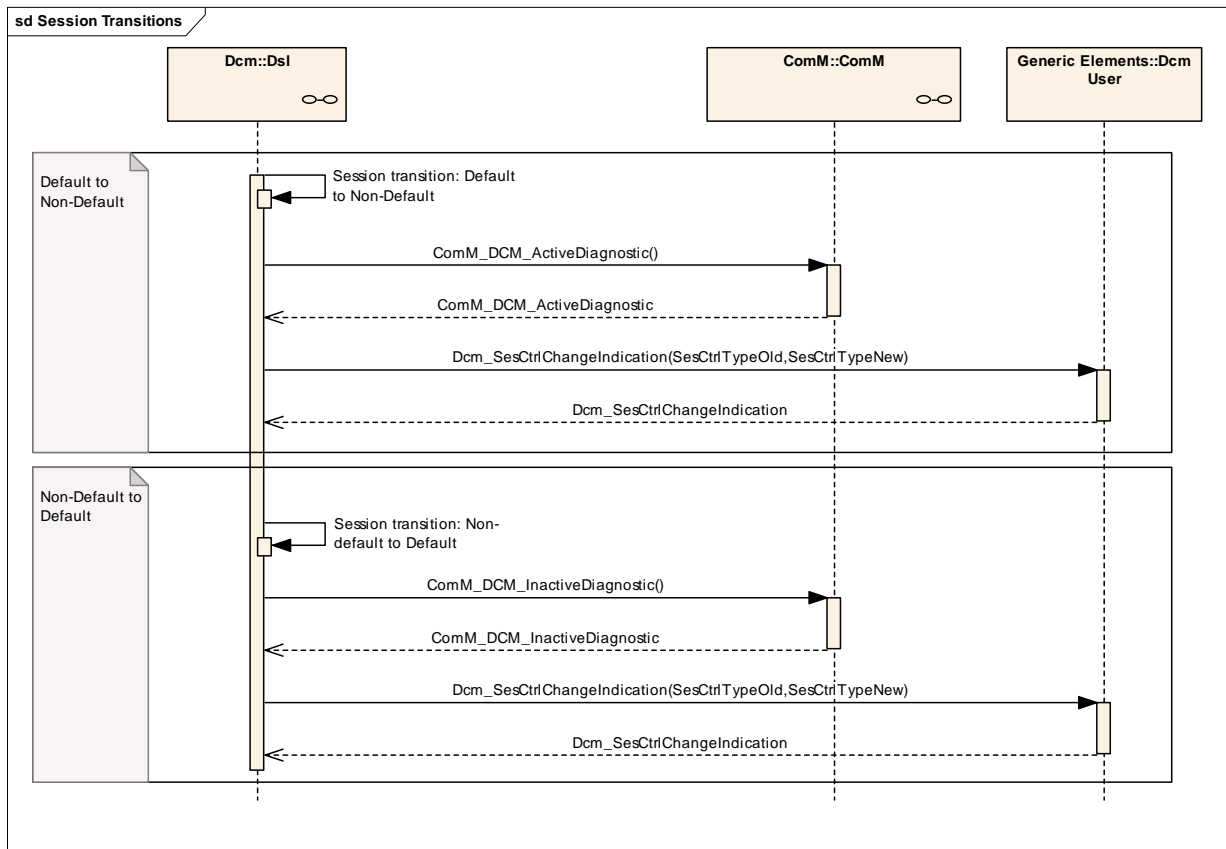
9.2.7.1 Handling in Default Session



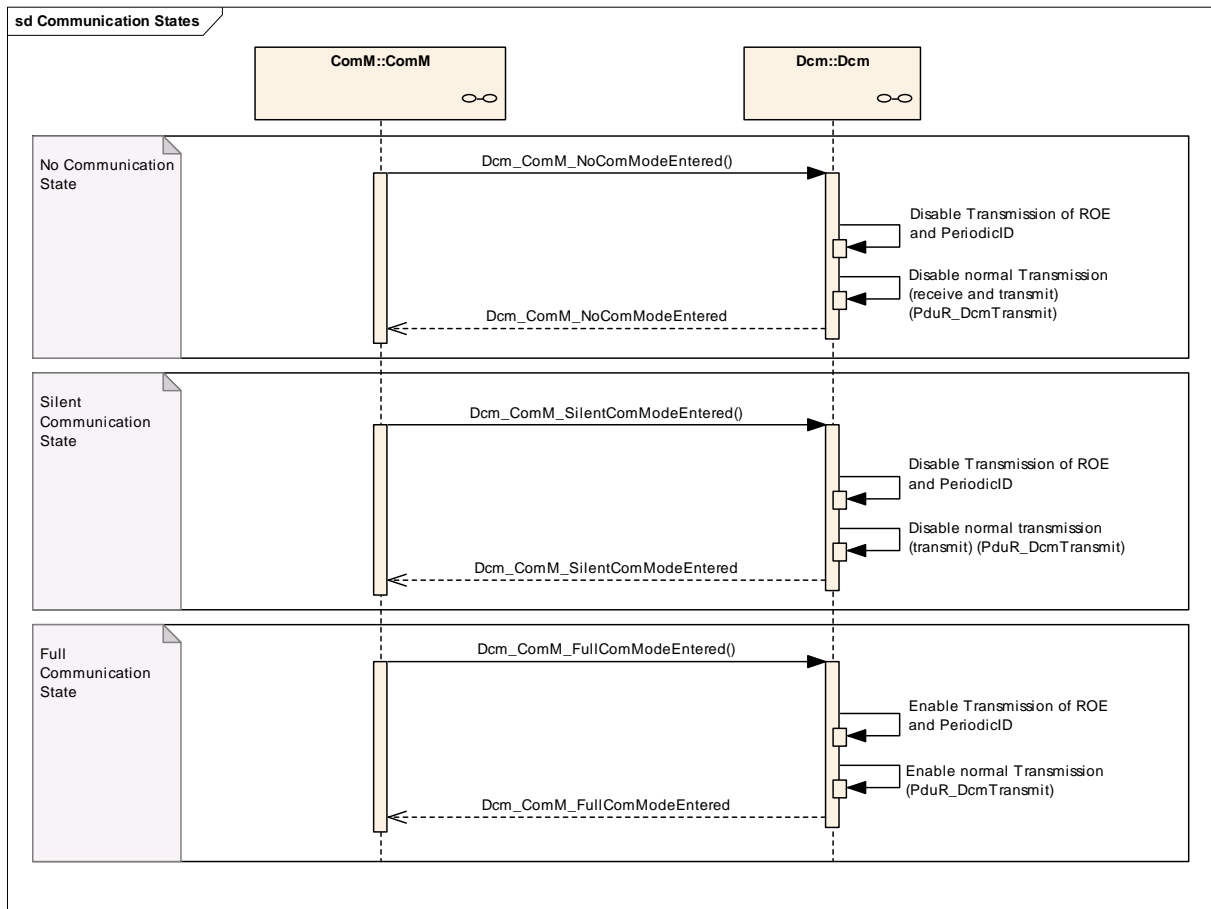
9.2.7.2 Handling in Non-Default Session



9.2.7.3 Session transitions

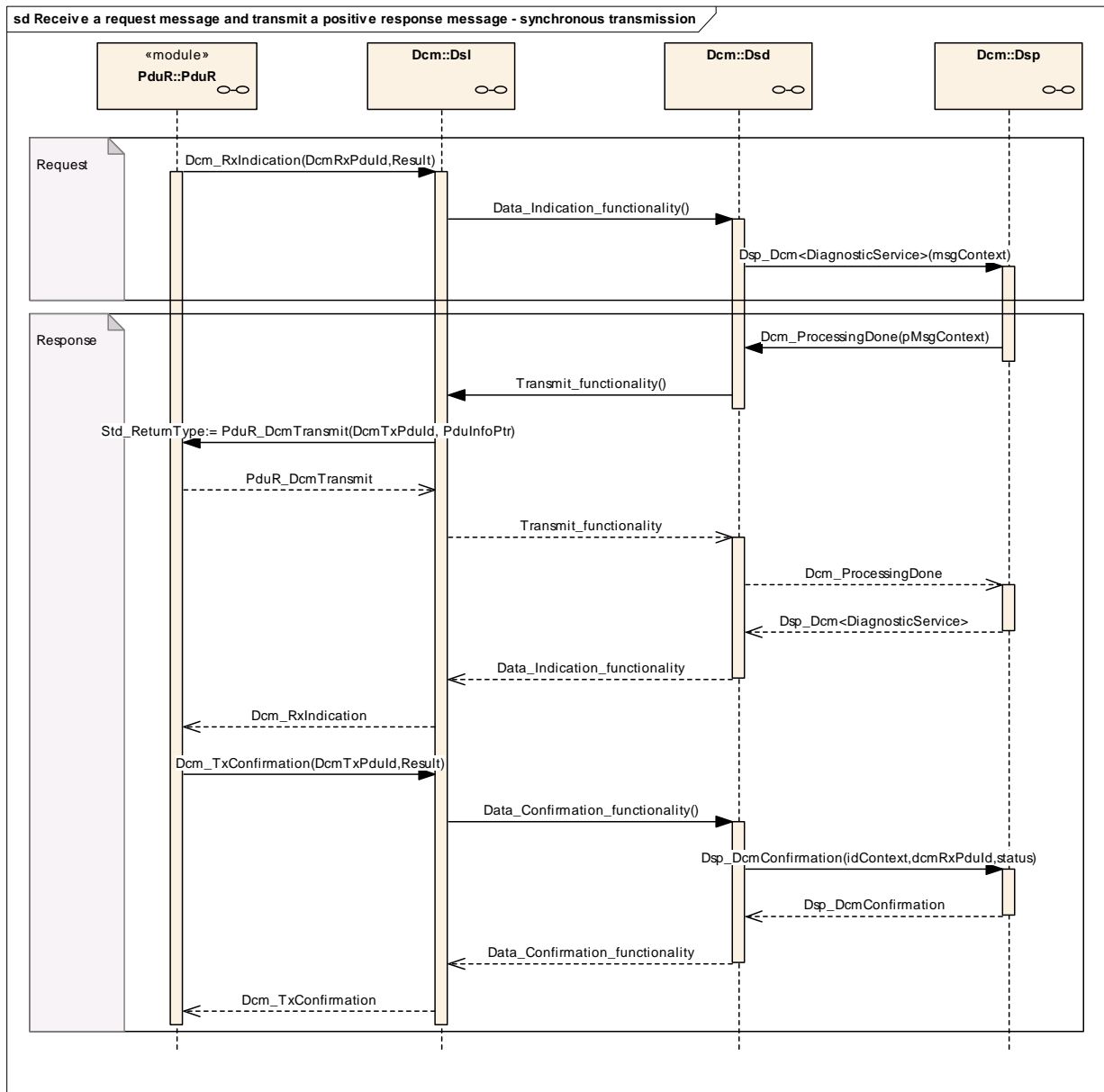


9.2.7.4 Communication States

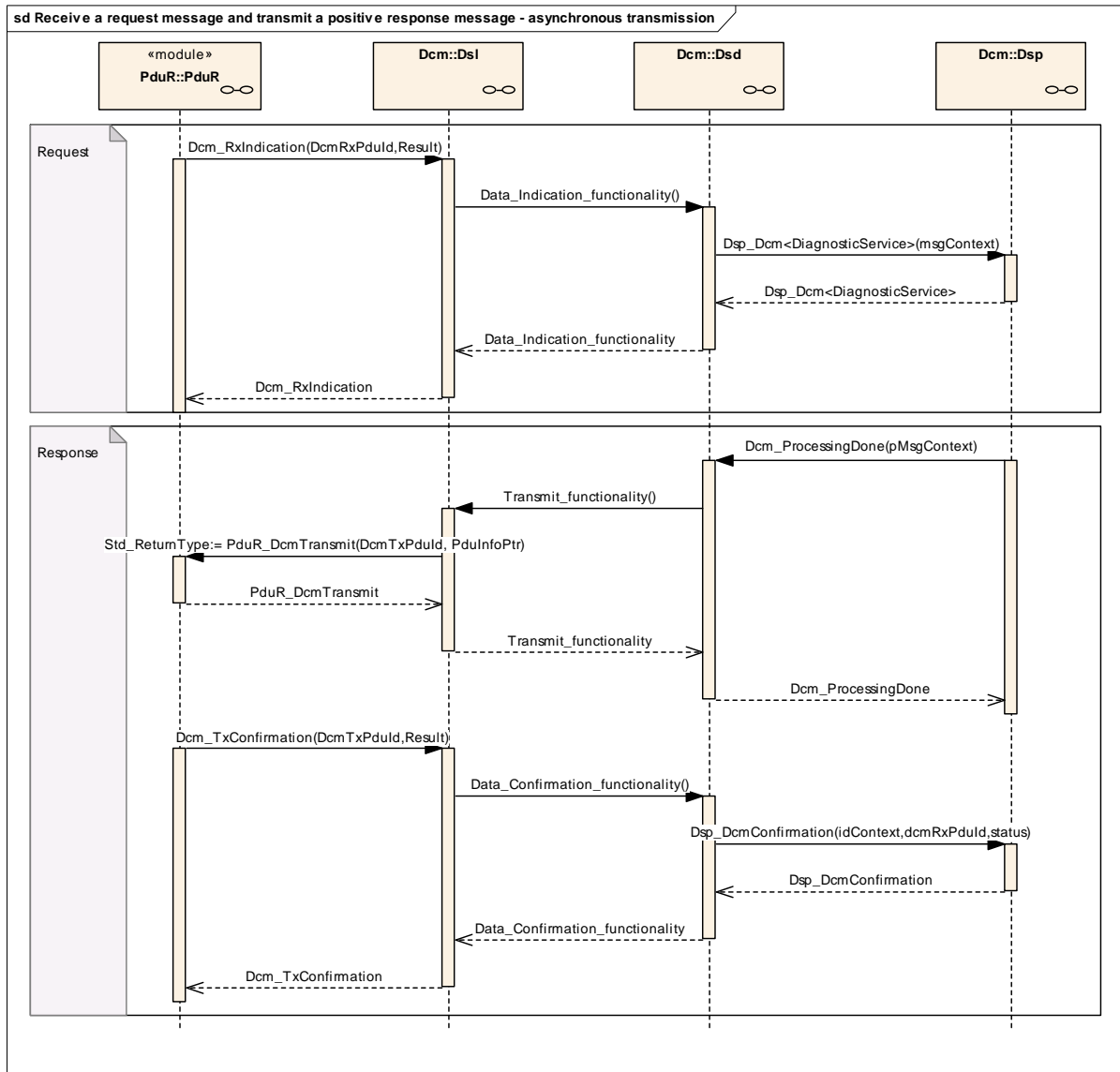


9.3 DSD (Diagnostic Service Dispatcher)

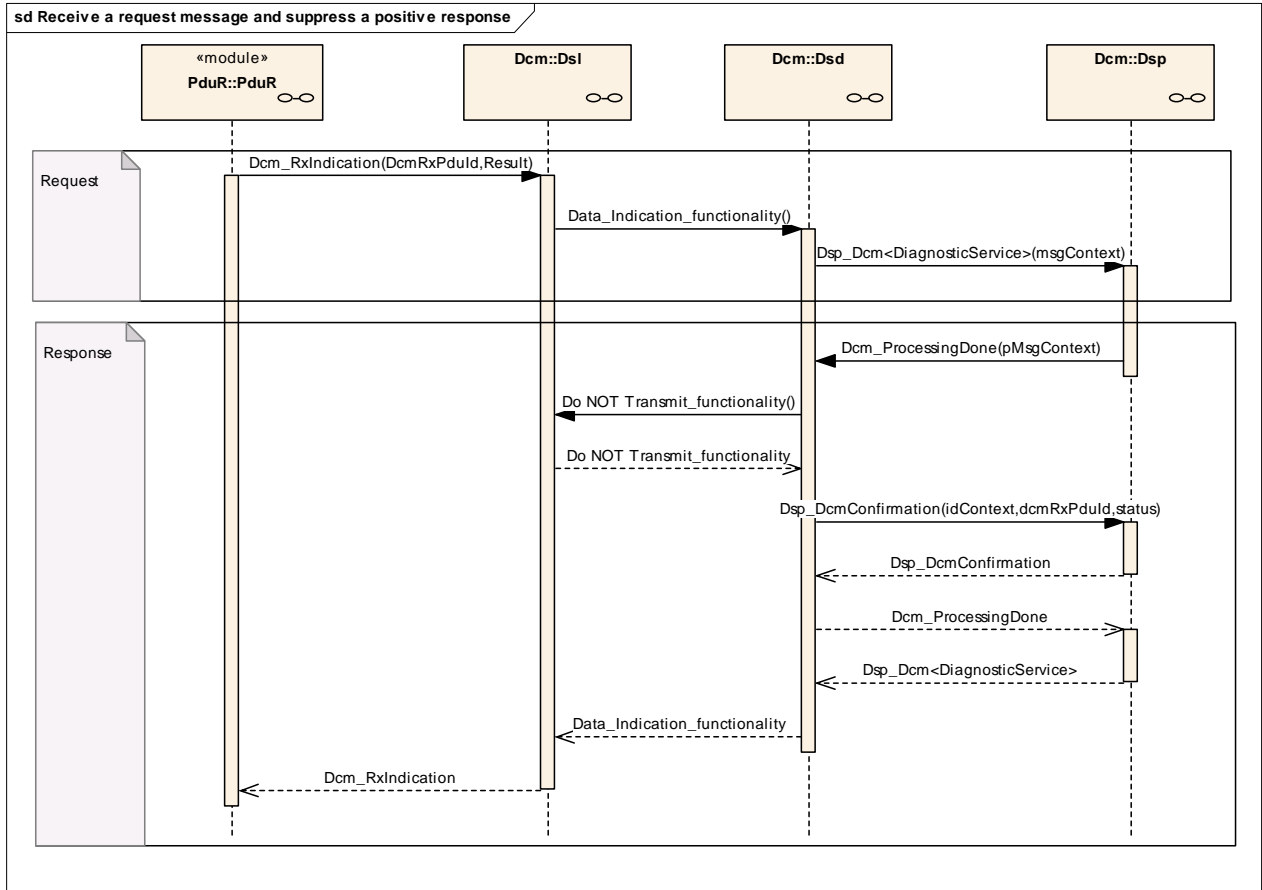
9.3.1 Receive a request message and transmit a positive response message – synchronous transmission



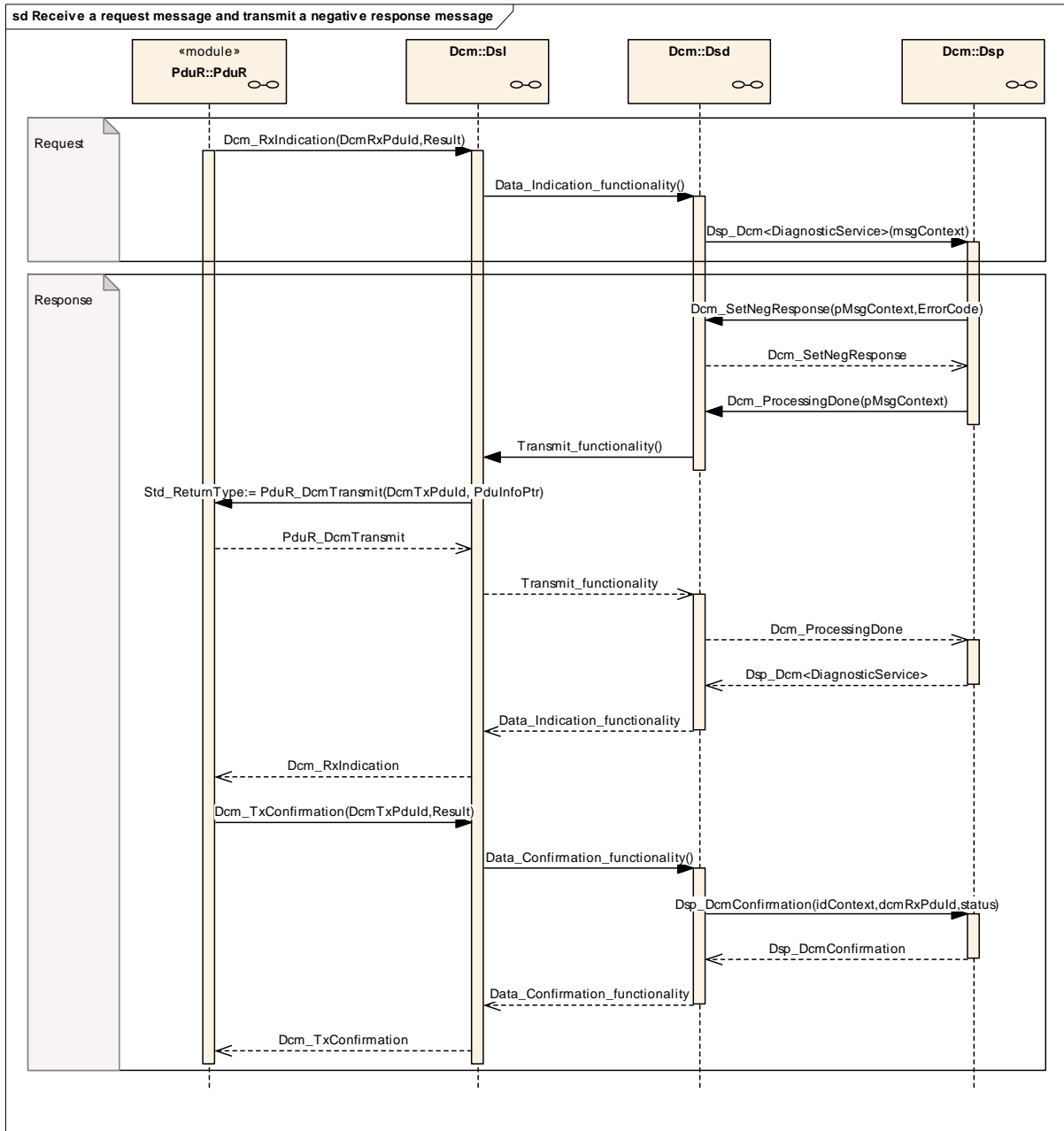
9.3.2 Receive a request message and transmit a positive response message – asynchronous transmission



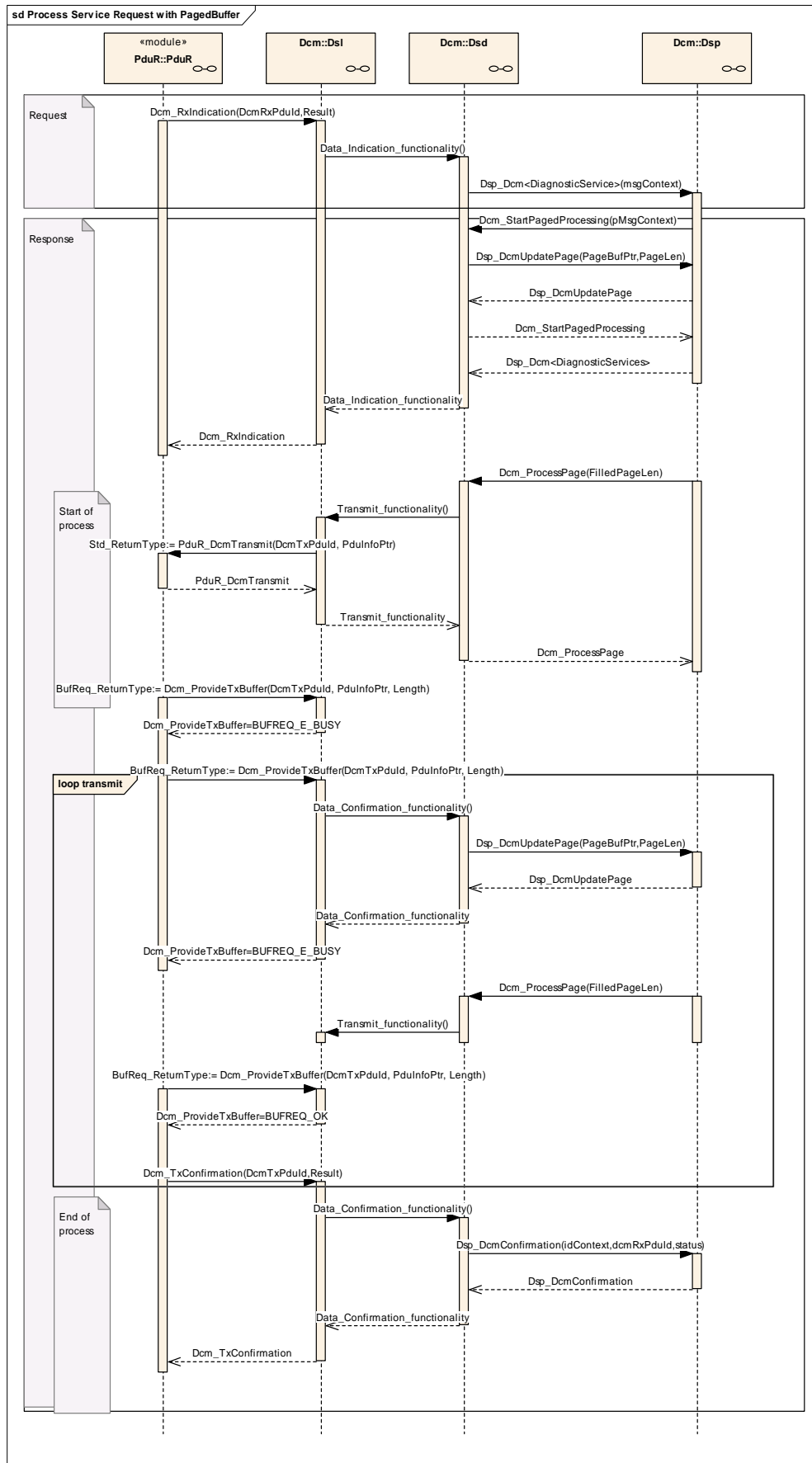
9.3.3 Receive a request message and suppression a positive response



9.3.4 Receive request message and transmit negative response message



9.3.5 Process Service Request with PagedBuffer



The following flow is processed in case no error occurs on application side:

Start of process:

4) Dcm_StartPagedProcessing()

With this API, the DSP gives the complete response length to DCM and starts PagedBuffer handling. This API starts no transmission!

5) Dsp_DcmUpdatePage()

DCM requests data to be transmitted.

6) Dcm_ProcessPage

With this API, the DSP requests transmission of the current page.

8) PduR_DcmTransmit()

DCM requests transmission to the lower layers

9) Dcm_ProvideTxBuffer()

The buffer is filled and DCM shall return "BUFREQ_OK"(10)

Start of loop:

11) Dcm_ProvideTxBuffer()

The Pdu Router requests the buffer but the buffer is not filled by the DSP.

12 + 13) Dsp_DcmUpdatePage

DCM requests the DSP to fill the next page

14) By returning "BUFREQ_E_BUSY" DCM indicates that the buffer has to be filled from the DSP.

15) Dcm_ProcessPage()

With this API, the DSP requests transmission of the current page.

17) Then, on the next call of Dcm_ProvideTxBuffer() the buffer is filled and DCM shall return "BUFREQ_OK" (18)

LOOP: The flow 10 to 18 is repeated as long data can be send.

End of process:

n-2 -> n) Dcm_TxConfirmation() When all data is send the Pdu Router indicates the send with a confirmation which is give to the DSP

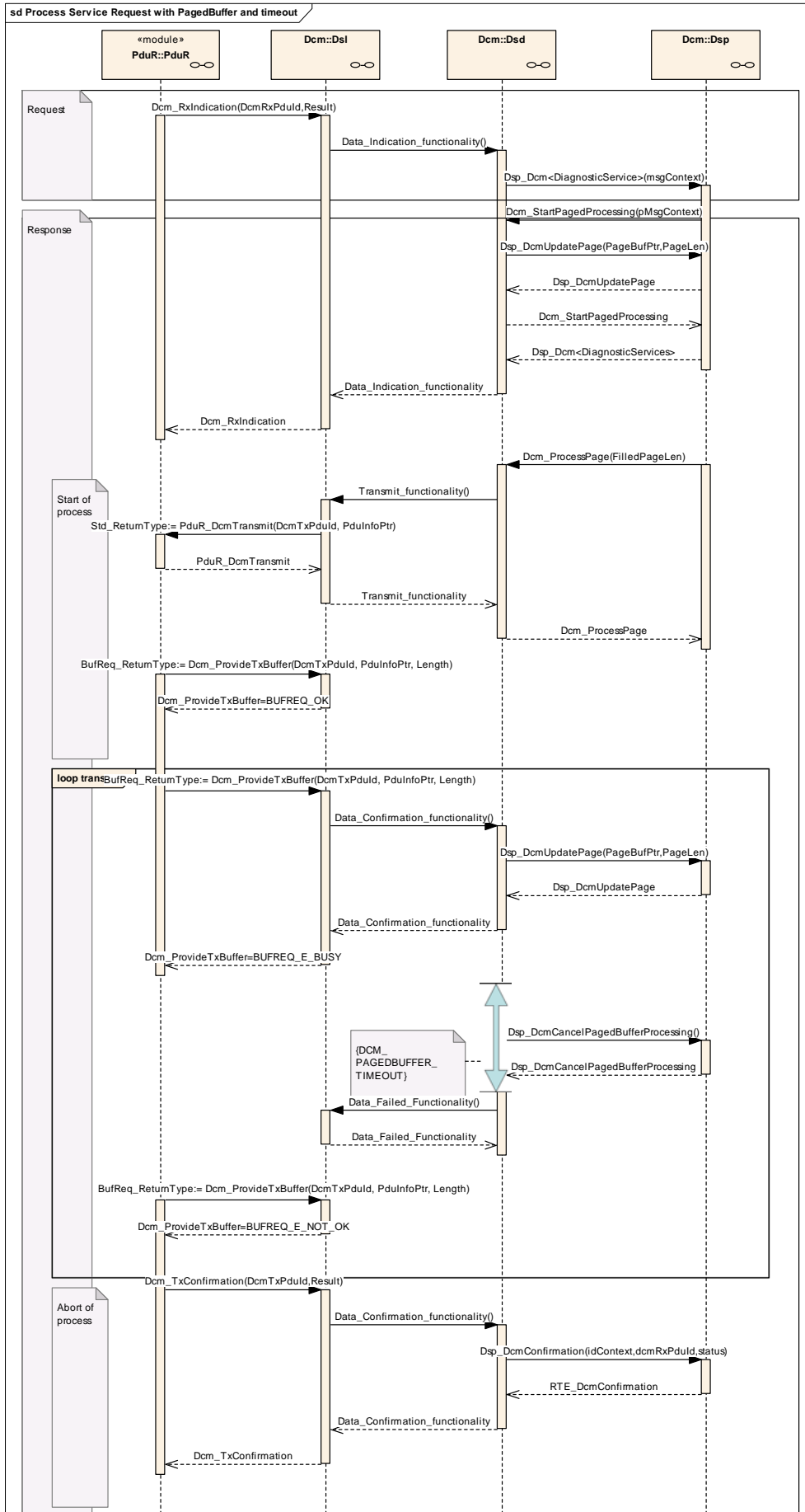
The APIs 4, 5 and 6 are needed only for PagedBuffer transmission

Page buffer timeout handling:

DCM react in the following described way, when DSP starts paged buffer handling, but is not able to process further on filling the response data. E.g. there are problems to access data from an EEPROM device.

When providing the Pagebuffer to DSP (13: Dsp_DcmUpdatePage()), DCM starts a timeout supervision. If timeout (value configured in **DcmPagedBufferTimeout**) occurs, before DSP requests next page (Dcm_ProcessPage()) following error handling is carried out in DCM:

- DCM stops further processing of paged buffer (item 15)
- DCM requests DSP (14: Dsp_DcmCancelPagedBufferProcessing()) to stop further processing of PagedBuffer
- DCM will cancel ongoing transmission in lower layers (done with return value BUFREQ_E_NOT_OK in next Dcm_ProvideTxBuffer() request, item 17).



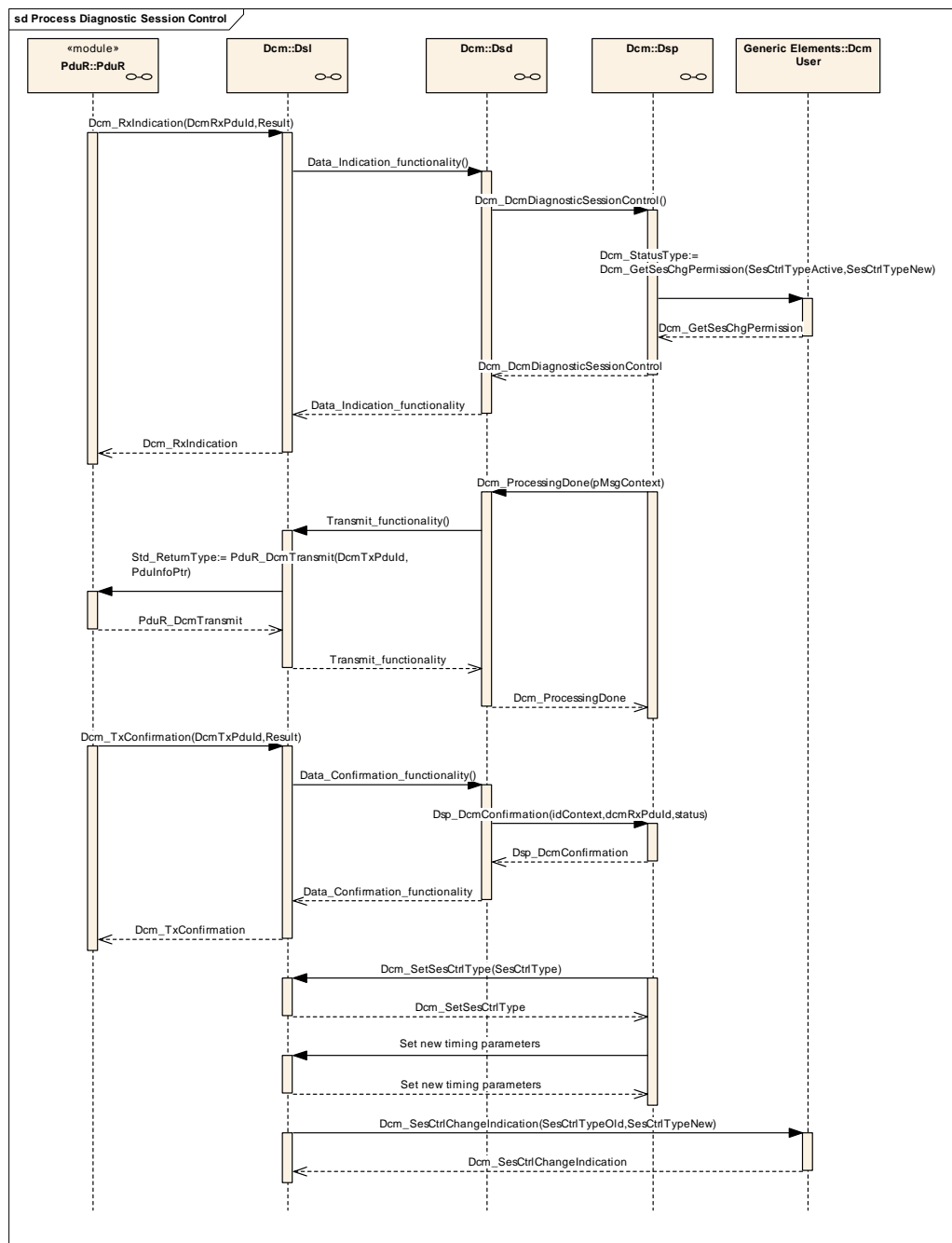
9.4 DSP (Diagnostic Service Processing)

9.4.1 Interface DSP – DEM (service 0x19, 0x14, 0x85)

Please refer to chapter 9. in [6].

9.4.2 Interface special services

9.4.2.1 Process Diagnostic Session Control

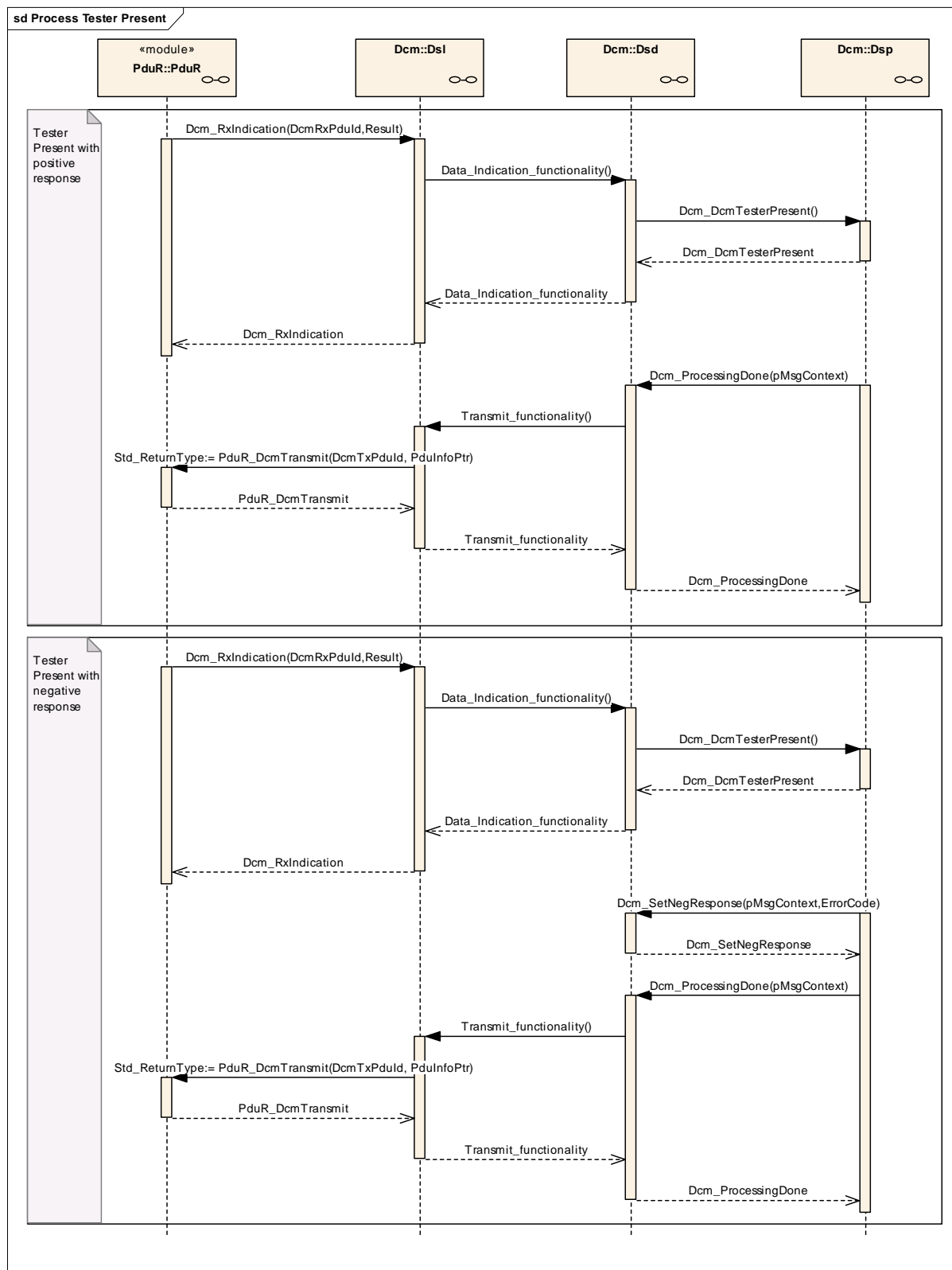


On a Diagnostic Session Control request from tester DSP requests the application to get the permission to change the session.

With the permission from the application a positive response is given to the PDU Router. In the data confirmation function the new session type and the new timing values are set.

DSL indicates the session change to the application (**Dcm_SesCtrlChangeIndication()**).

9.4.2.2 Process Tester Present



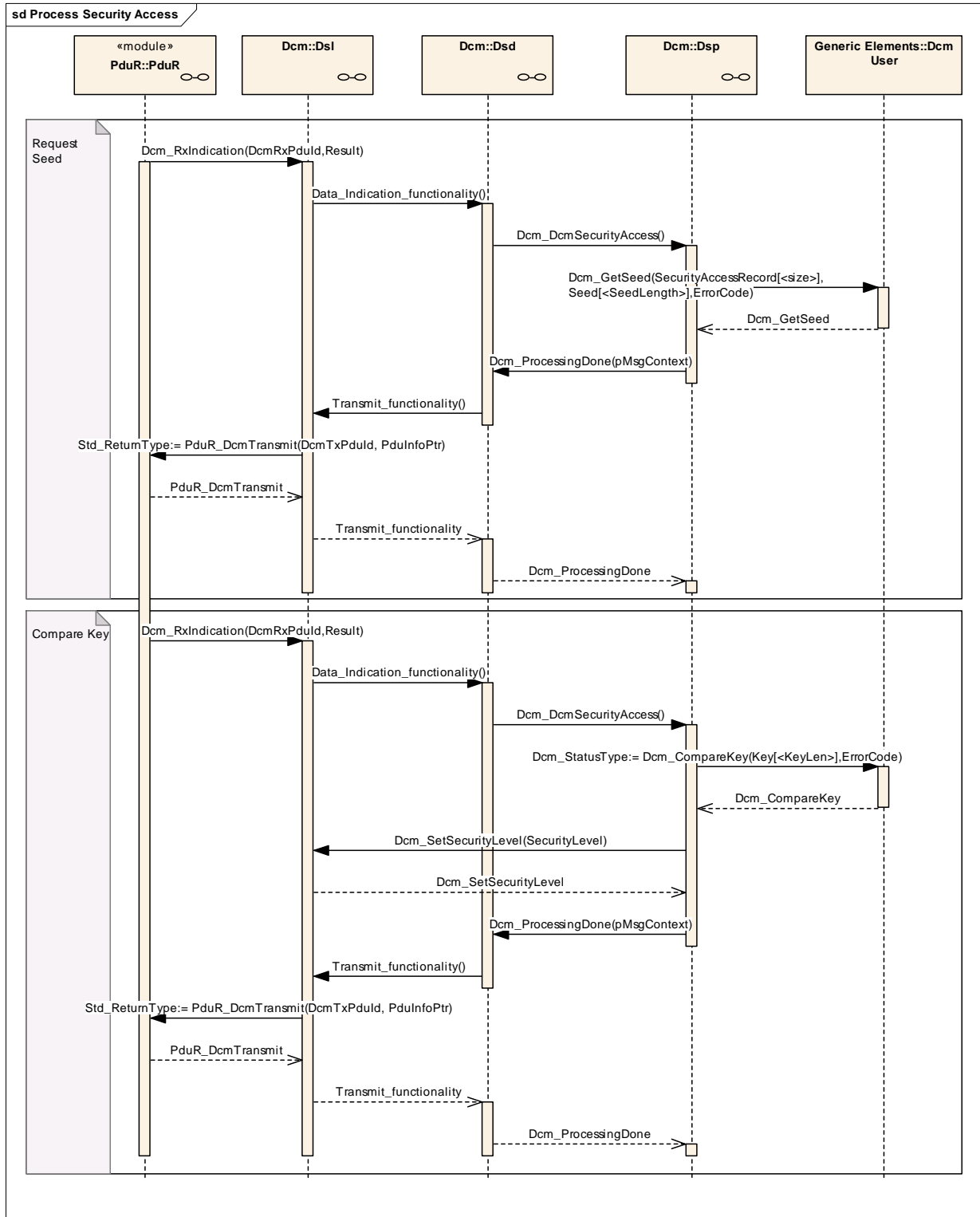
Above sequence diagram shows processing of TesterPresent commands, which are not of type functional addressed with Subfunction 0x80. These TesterPresent commands are interpreted in the DSL (more details can be found in chapter 7.2.1.4.2.2 Concurrent “TesterPresent” (“keep alive logic”))

All the other TesterPresent commands are processed in the following way:

On a command TesterPresent the DSD calls the DSP with the function **Dcm_DcmTesterPresent()**.

The sequence chart also shows the case when an error occurs and a negative response is sent.

9.4.2.3 Process Security Access



To get the security access, DSD has to call DSP to get the seed value from the application. If no error is detected, the seed value is sent in the positive response.

In second step DSP gets the by tester calculated key and requests the application to compare this key. If no error occurs, the new access type is set in the DSL and a positive response is send.

10 Configuration specification

Dcm067: In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals.

Chapters 10.2, 10.3, 10.4, 10.5 specifies the structure (containers) and the parameters of the module DCM.

Chapter 10.6 specifies published information of the module DCM.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [5]
- AUTOSAR ECU Configuration Specification [6]. This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Dcm011: Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

Pre-compile time specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	Description
x	The configuration parameter shall be of configuration class <i>Pre-compile time</i> .
--	The configuration parameter shall never be of configuration class <i>Pre-compile time</i> .

Link time specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	Description
x	The configuration parameter shall be of configuration class <i>Link time</i> .

--	The configuration parameter shall never be of configuration class <i>Link time</i> .
----	--

Post Build specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	Description
x	The configuration parameter shall be of configuration class <i>Post Build</i> and no specific implementation is required.
L	<i>Loadable</i> - the configuration parameter shall be of configuration class <i>Post Build</i> and only one configuration parameter set resides in the ECU.
M	<i>Multiple</i> - the configuration parameter shall be of configuration class <i>Post Build</i> and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class <i>Post Build</i> .

10.1.2 Variants

The DCM shall have the following variants:

Variant	Description
Variant A	This variant is limited to pre-compile configuration parameters only..
Variant B	This variant is limited to link time configuration parameters only. Variant B shall be used when the DCM implementation will be delivered in object code only.
Variant C	This variant allows a mix of pre-compile time- and post build time - configuration parameters. In this variant the pre-compile Parameter shall be used to enable or disable the functionality (e.g. ROE transmission DCM_ROE_ENABLED). With the post build parameter the functionality shall be configured (e.g. DCM_ROE_TRANS_TYPE).
Mandatory [pre-compile parameter] for all variants	Please note: This pre-compile configuration parameters are mandatory for all Variant. The pre-compile Parameter shall be used to enable or disable DCM functionality (e.g. ROE transmission DCM_ROE_ENABLED) or are DCM global configuration data which shall be configured at pre compile time (e.g. memory influence).

10.1.3 Containers

Dcm057: Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

Dcm058: Containers shall have names, which indicate what kinds of parameters are handled inside.

10.2 DCM global Configurations

Dcm064:

10.2.1 Dcm

Module Name	Dcm
Module Description	Configuration of the Dcm (Diagnostic Communications Manager) module.

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDsp	1	These parameters apply to Diagnostic Service Processing. There will always be one set of these parameters per DCM.
DcmGeneral	1	This container contains the configuration (parameters) for Component wide parameters
DcmPageBufferCfg	1	This container contains the configuration (parameters) for Page Buffer handling
DcmDsd	1	These parameters apply to Diagnostic Service Dispatcher. All parameters for all service dispatchers are included in this one configuration container. upperMultiplicity: Service dispatcher is multiple times configurable in this one configuration container. lowerMultiplicity: There must always be one service dispatcher in a DCM.
DcmDsl	1	These parameters apply to a Diagnostic Session Layer. There may be a parameter set (DSL Configuration) per protocol. upperMultiplicity: Each DCM configuration must have exactly one DSL configuration. lowerMultiplicity: Each DCM configuration must have exactly one DSL configuration.

10.2.1.1 COMPONENT_WIDE_PARAMETERS

SWS Item	DCM075, DCM076 :		
Container Name	DcmGeneral{COMPONENT_WIDE_PARAMETERS}		
Description	This container contains the configuration (parameters) for Component wide parameters		
Configuration Parameters			

SWS Item	DCM075, DCM076 :		
Name	DcmVersionInfoApi {DCM_VERSION_INFO_API}		
Description	Preprocessor switch to enable or disable the output Version info of the functionality.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	DCM075, DCM076 :		
Name	DcmRequestIndicationEnabled {DCM_REQUEST_INDICATION_ENABLED}		
Description	Allows to enable or disable the requested indication mechanism.		
Multiplicity	1		

Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	DCM075, DCM076 :		
Name	DcmDevErrorDetect {DCM_DEV_ERROR_DETECT}		
Description	Preprocessor switch to enable or disable the Development Error Detection (DET) mechanism.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	DCM075, DCM076 :		
Name	DcmTaskTime {DCM_TASK_TIME}		
Description	Allow to configure the time for the periodic cyclic task (in ms). Please note: This configuration value shall be equal to the value in the ScheduleManager module. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM. min: A negative value is not allowed. upperMultiplicity: Exactly one TaskTime must be specified per configuration. lowerMultiplicity: Exactly one TaskTime must be specified per configuration.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.1.2 PAGE_BUFFER_CFG

SWS Item	DCM068 :		
Container Name	DcmPageBufferCfg{PAGE_BUFFER_CFG}		
Description	This container contains the configuration (parameters) for Page Buffer handling		
Configuration Parameters			

SWS Item	DCM068 :		
Name	DcmPagedBufferTimeout {DCM_PAGEDBUFFER_TIMEOUT}		
Description	Allow to configure the Timeout (in ms) towards the application for filling the next page. This parameter is only relevant if the Page Buffer handling is enabled. (DcmPagedBufferEnabled = TRUE) Defined in the DCM SWS as		

	uint16, this parameter contains a value in milliseconds. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM. min: A negative value is not allowed. upperMultiplicity: Exactly one Timeout must be specified per configuration. lowerMultiplicity: Exactly one Timeout must be specified per configuration.		
Multiplicity	0..1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: dependency: DcmPagedBufferEnabled		ECU

SWS Item	DCM068 :		
Name	DcmPagedBufferEnabled {DCM_PAGEDBUFFER_ENABLED}		
Description	Allow to enable or disable the Page buffer mechanism. true = Page buffer handling enabled false = Page Buffer handling disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.3 DSD (Diagnostic Service Dispatcher)

10.3.1 DcmDsdServiceTable

SWS Item	Dcm071 :		
Container Name	DcmDsdServiceTable{SERVICE_IDENTIFIER_TABLE}		
Description	This container contains the configuration (DSD parameters) for Service Identifier Table.		
Configuration Parameters			

SWS Item	Dcm071 :		
Name	DcmDsdSidTabId {DCM_SIDTAB_ID}		
Description	Due the fact of using one or more service tables the member of the Service Identifier Table includes a unique id for the Service Identifier Table.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME

	Post-build time	--	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDsdService	1..*	This container contains the configuration (DSD parameters) for Service.

10.3.2 DcmDsdService

SWS Item	--
Container Name	DcmDsdService
Description	This container contains the configuration (DSD parameters) for Service.
Configuration Parameters	

SWS Item	Dcm071 :		
Name	DcmDsdSidTabSubfuncAvial {DCM_SIDTAB_SUBFUNC_AVAIL}		
Description	Information whether the DcmDsdSidTabServiceId includes Sub functions or not. Used for the Handling of "suppressPosRspMsgIndicationBit" (chapter 7.2.2.4) ISO14229-1 can be referenced here, as this specification gives fix definition, if an SID includes Subfunction or not. true = sub-function available false = sub-function not available		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: dependency: DcmDsdSidTabServiceId		ECU

SWS Item	Dcm071 :		
Name	DcmDsdSidFunction {DCM_SIDTAB_FUNCTIONPOINTER}		
Description	Function Pointer for the callback function of the DSP or/and the Autosar SW component for the particular DcmDsdSidTabServiceId. Please refer to the ISO 14229-1, ISO 15031-5 and "Distribution of Diagnostic Message to data processor" (chapter 7.2.2.4). Type: void XXX_Dcm<DiagnosticService>(Dcm_MsgContextType*)		
Multiplicity	1		
Type	FunctionNameDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: dependency: DcmDsdSidTabServiceId		ECU

SWS Item	--
Name	DcmDsdSidTabSidInitFunction
Description	Function Pointer for the init callback function of the DSP or/and the Autosar SW component for the particular DcmDsdSidTabServiceId. Please refer to the ISO 14229-1, ISO 15031-5 and "Distribution of Diagnostic Message to data processor" (chapter 7.2.2.4). All the configured service

	init functions will be processed, when starting the diagnostic protocol (chapter 7.2.1.4.2.14) Type: void XXX_DcmInit<DiagnosticService>(void)		
Multiplicity	1		
Type	FunctionNameDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: dependency: DcmDsdSidTabServiceId		ECU

SWS Item	Dcm071 :		
Name	DcmDsdSidTabServiceId {DCM_SIDTAB_SERVICEID}		
Description	Id of the Service identifier in hex. The possible Service identifier are predefined in the ISO 14229-1 and ISO 15031-5 and in Table 5 and Table 6.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	Dcm071 :		
Name	DcmDsdSidTabSessionLevelRef {DCM_SIDTAB_SESSION_LEVEL_ROW}		
Description	Link to the Session Control needed for execution of the DcmDsdService. Please refer to the ISO 14229-1, ISO 15031-5 and "Verification of the Diagnostic Session" (chapter 7.2.2.4). For DcmDspSessionLevel please refer to chapter 10.5 DSP (Diagnostic Service Processing) Please note, that it shall be provided to configure several DcmDsdSidTabSessionLevelRef per DcmDsdService.		
Multiplicity	1..*		
Type	Reference to DcmDspSessionRow		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	Dcm071 :		
Name	DcmDsdSidTabSecurityLevelRef {DCM_SIDTAB_SEC_LEVEL_ROW}		
Description	Link to the Security Access Levels needed for execution of the DcmDsdService. Please refer to the ISO 14229-1, ISO 15031-5 and "Verification of the Security Access levels" (chapter 7.2.2.4). For DcmDspSecurityLevel please refer to chapter 10.5 DSP (Diagnostic Service Processing)Please note, that it shall be provided to configure several DcmDsdSidTabSecurityLevelRef per DcmDsdService.		
Multiplicity	1..*		
Type	Reference to DcmDspSecurityRow		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-

			POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.3.3 DcmDsd

SWS Item	--
Container Name	DcmDsd
Description	These parameters apply to Diagnostic Service Dispatcher. All parameters for all service dispatchers are included in this one configuration container. upperMultiplicity: Service dispatcher is multiple times configurable in this one configuration container. lowerMultiplicity: There must always be one service dispatcher in a DCM.
Configuration Parameters	

Included Containers

Container Name	Multiplicity	Scope / Dependency
DcmDsdServiceTable	1..256	This container contains the configuration (DSD parameters) for Service Identifier Table.

10.4 DSL (Diagnostic Session Layer)

SWS Item	--
Container Name	DcmDsl
Description	These parameters apply to a Diagnostic Session Layer. There may be a parameter set (DSL Configuration) per protocol. upperMultiplicity: Each DCM configuration must have exactly one DSL configuration. lowerMultiplicity: Each DCM configuration must have exactly one DSL configuration.
Configuration Parameters	

Included Containers

Container Name	Multiplicity	Scope / Dependency
DcmDslProtocol	1	This container contains the configuration (parameters) for the protocol configuration (for each protocol) The following parameters needs to be configured per protocol.
DcmDslProtocolTiming	1	This container contains the configuration (parameters) for Protocol timing. This container contains rows of DcmDslProtocolTimingRow containers. Hint: DcmDslProtocolTiming is only relevant in case of using the UDS service AccessTimingParameter.
DcmDslDiagResp	1	This container contains the configuration (parameters) for the ResponsePending handling
DcmDslBuffer	1..256	This container contains the configuration (parameters) for the diagnostic buffer.

10.4.1 DcmDslBuffer

SWS Item	DCM032 :
Container Name	DcmDslBuffer{DIAGNOSTIC_BUFFER_CFG}
Description	This container contains the configuration (parameters) for the diagnostic buffer.
Configuration Parameters	

SWS Item	DCM032 :		
Name	DcmDslBufferID {DCM_BUFFER_ID}		
Description	Identifier of Buffer.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	DCM032 :		
Name	DcmDslBufferSize {DCM_BUFFER_SIZE}		
Description	Size of Diagnostic Buffer (in Bytes) For a linear buffer: size of the buffer shall be as large as the longest message (request or response) For a paged buffer (only Tx possible): size has impacts on the application performance Please note: max. range is the valid range for a CAN network. We assume a FlexRay (or other networks) implementation will work with this range (and the page buffer mechanism) without any problems.		
Multiplicity	1		
Type	IntegerParamDef		
Range	8 .. 4095		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.4.2 DcmDslConnection

SWS Item	--
Choice Container Name	DcmDslConnection
Description	This container contains links between Diagnostic Protocol (=DcmDslProtocolRow) and the according Rx or Tx channel. Because of the usecase to allow more then one diagnostic tester (using different CAN channels) it is necessary to allow configuration of multiple DcmDslConnections.

Container Choices		
Container Name	Multiplicity	Scope / Dependency

DcmDslMainConnection	1	This container contains configuration for Diagnostic Main Connection (DcmDslProtocolTx, DcmDslProtocolRx). In addition it contains links to the ROE connection (DcmDslROEConnectionRef) and the Periodic Transmission Connection (DcmDslPeriodicTransmissionConRef) relevant for protocols with DcmDslProtocolTransType = TYPE2.
DcmDslPeriodicTransmission	0..1	This container contains the configuration (parameters) for Periodic Transmission service. Hint: Periodic Transmission request comes via DcmDslMainConnection and PeriodicTransmission Event response is given via DcmDslPeriodicTransmission.
DcmDslResponseOnEvent	0..1	This container contains the configuration (parameters) for ResponseOnEvent service. Hint: ROE service request is received via DcmDslMainConnection and that Event response is given via DcmDslResponseOnEvent

10.4.3 DcmPageBufferCfg

SWS Item	DCM068 :		
Container Name	DcmPageBufferCfg{PAGE_BUFFER_CFG}		
Description	This container contains the configuration (parameters) for Page Buffer handling		
Configuration Parameters			

SWS Item	DCM068 :		
Name	DcmPagedBufferTimeout {DCM_PAGEDBUFFER_TIMEOUT}		
Description	Allow to configure the Timeout (in ms) towards the application for filling the next page. This parameter is only relevant if the Page Buffer handling is enabled. (DcmPagedBufferEnabled = TRUE) Defined in the DCM SWS as uint16, this parameter contains a value in milliseconds. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM. min: A negative value is not allowed. upperMultiplicity: Exactly one Timeout must be specified per configuration. lowerMultiplicity: Exactly one Timeout must be specified per configuration.		
Multiplicity	0..1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: dependency: DcmPagedBufferEnabled		ECU

SWS Item	DCM068 :		
Name	DcmPagedBufferEnabled {DCM_PAGEDBUFFER_ENABLED}		
Description	Allow to enable or disable the Page buffer mechanism. true = Page buffer handling enabled false = Page Buffer handling disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: ECU
---------------------------	------------

No Included Containers

10.4.4 DcmDslMainConnection

SWS Item	--
Container Name	DcmDslMainConnection{DIAGNOSTIC_CONNECTION_TABLE}
Description	This container contains configuration for Diagnostic Main Connection (DcmDslProtocolTx, DcmDslProtocolRx). In addition it contains links to the ROE connection (DcmDslROEConnectionRef) and the Periodic Transmission Connection (DcmDslPeriodicTransmissionConRef) relevant for protocols with DcmDslProtocolTransType = TYPE2.
Configuration Parameters	

SWS Item	--									
Name	DcmDslPeriodicTransmissionConRef									
Description	Configures the link to the connection of PeriodicTransmission protocol for the processing of the PeriodicTransmission events.									
Multiplicity	0..1									
Type	Reference to DcmDslPeriodicTransmission									
ConfigurationClass	<table border="1"> <tr> <td>Pre-compile time</td> <td>X</td> <td>VARIANT-PRE-COMPILE</td> </tr> <tr> <td>Link time</td> <td>X</td> <td>VARIANT-LINK-TIME</td> </tr> <tr> <td>Post-build time</td> <td>L</td> <td>VARIANT-POST-BUILD</td> </tr> </table>	Pre-compile time	X	VARIANT-PRE-COMPILE	Link time	X	VARIANT-LINK-TIME	Post-build time	L	VARIANT-POST-BUILD
Pre-compile time	X	VARIANT-PRE-COMPILE								
Link time	X	VARIANT-LINK-TIME								
Post-build time	L	VARIANT-POST-BUILD								
Scope / Dependency	scope: ECU									

SWS Item	--									
Name	DcmDslROEConnectionRef									
Description	Configures the link to the connection of ROE protocol for processing of the ROE events.									
Multiplicity	0..1									
Type	Reference to DcmDslResponseOnEvent									
ConfigurationClass	<table border="1"> <tr> <td>Pre-compile time</td> <td>X</td> <td>VARIANT-PRE-COMPILE</td> </tr> <tr> <td>Link time</td> <td>X</td> <td>VARIANT-LINK-TIME</td> </tr> <tr> <td>Post-build time</td> <td>L</td> <td>VARIANT-POST-BUILD</td> </tr> </table>	Pre-compile time	X	VARIANT-PRE-COMPILE	Link time	X	VARIANT-LINK-TIME	Post-build time	L	VARIANT-POST-BUILD
Pre-compile time	X	VARIANT-PRE-COMPILE								
Link time	X	VARIANT-LINK-TIME								
Post-build time	L	VARIANT-POST-BUILD								
Scope / Dependency	scope: ECU									

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDslProtocolTx	1	This container contains the configuration (parameters) for the protocol configuration of TX channel (for each protocol) The included parameters needs to be configured per protocol.
DcmDslProtocolRx	1..*	This container contains the configuration (parameters) for the protocol configuration of RX channel (for each protocol) The following parameters needs to be configured per protocol. Please keep in mind, that the parameter DcmDslProtocolRxPduld can be given several times per protocol and that the combination from DcmDslProtocolRxPduld and DcmDslProtocolRxAddrType is unique. Only one physical protocol is allowed per connection. upperMultiplicity: More than one receive PDU ID may be configured for reception (e.g. one for func requests, one for phys requests). lowerMultiplicity: At least one receive PDU ID configuration must be provided per protocol.

10.4.5 DcmDslPeriodicTransmission

SWS Item	Dcm070 :
Container Name	DcmDslPeriodicTransmission{PERIODIC_TRANSMISSION_PARAMETERS}
Description	This container contains the configuration (parameters) for Periodic Transmission service. Hint: Periodic Transmission request comes via DcmDslMainConnection and PeriodicTransmission Event response is given via DcmDslPeriodicTransmission.
Configuration Parameters	

SWS Item	Dcm070 :		
Name	DcmDslPeriodicTxPduRef {DCM_PERIODIC_TRANS_DCMTXPDUID}		
Description	This is the reference to the PDU ID to be used by this DCM when sending Periodic Transmissions. It is only needed for Type2 Periodic Transmissions configurations. upperMultiplicity: Exactly one DcmDslPeriodicTxPduRef must be defined if Periodic Transmission is enabled and TYPE2 Periodic Transmissions is configured. lowerMultiplicity: DcmDslPeriodicTxPduRef does not need to be defined if Periodic Transmission is not enabled or TYPE1 Periodic Transmissions is configured..		
Multiplicity	0..*		
Type	Reference to Pdu		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: DcmDslPeriodicTransEnabled, DcmDslPeriodicTransType		

No Included Containers

10.4.6 DcmDslProtocol

SWS Item	--
Container Name	DcmDslProtocol{DIAGNOSTIC_PROTOCOL_TABLE}
Description	This container contains the configuration (parameters) for the protocol configuration (for each protocol) The following parameters needs to be configured per protocol.
Configuration Parameters	

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDslProtocolRow	1..*	Definition of a single Row of configuration for the protocol configuration (for each protocol)

10.4.7 DcmDslProtocolRow

SWS Item	--
Container Name	DcmDslProtocolRow{DIAGNOSTIC_PROTOCOL_TABLE_ROW}
Description	Definition of a single Row of configuration for the protocol configuration (for each protocol)
Configuration Parameters	

SWS Item	--
-----------------	----

Name	DcmDslProtocolTransType		
Description	Selects the transmission type for protocol.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	Type1	Messages on the DcmTxPduld already used for normal diagnostic responses. The outgoing messages must be synchronized with 'normal outgoing messages', which have a higher priority.	
	Type2	Messages on a separate DcmTxPduld.	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: dependency: DcmDslProtocol.DcmDslProtocolRow.DcmDslProtocolID		ECU

SWS Item	--		
Name	DcmDslProtocolPreemptTimeout {DCM_PROTOCOL_PREEMPT_TIMEOUT}		
Description	This is the value for the timeout (in milliseconds) of preempting protocol until protocol needs to be started. This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM. min: A negative value is not allowed. upperMultiplicity: Exactly one DcmDslProtocolPreemptTimeout value per DSL protocol timing structure. lowerMultiplicity: Exactly one DcmDslProtocolPreemptTimeout value per DSL timing structure. origin: Standard AUTOSAR configuration parameter.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: dependency: DcmDslProtocolID		ECU

SWS Item	--		
Name	DcmDslProtocolID {DCM_PROTOCOL_ID}		
Description	The SWS for DCM defines this as enum type of maxsize uint8 with values {OBD, UDS, ...}. It is effectively the name of the diagnostic protocol type for the DCM DSL protocol that is being configured. 0x00 : DCM_OBD_ON_CAN OBD on CAN (ISO15765-4; ISO15031-5) 0x01 : DCM_uds_on_can UDS on CAN (ISO15765-3; ISO14229-1) 0x02 : DCM_uds_on_flexray UDS on FlexRay (Manufacturer specific; ISO14229-1) 0x03 : DCM_roe_on_can 0x04 : DCM_roe_on_flexray 0x05 : DCM_periodic_on_can 0x06 : DCM_periodic_on_flexray Implementation Type: Dcm_ProtocolType 0x07..0xEF : Reserved for further AUTOSAR implementation 0xF0..0xFF : Reserved for SW supplier specific		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE

	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: dependency: DcmDslProtocolRxAddrType		ECU

SWS Item	--		
Name	DcmDslProtocolPriority {DCM_PROTOCOL_PRIO}		
Description	The SWS for DCM defines this item as uint8. This is the protocol priority that is used during protocol preemption handling. 0 = Highest priority (protocol may not be preempted by other protocols) 1, 2, 3... = Reducing priority. Protocol may be preempted by other protocols with lower priority value. upperMultiplicity: Exactly one priority must be provided per Tx Protocol. lowerMultiplicity: Exactly one priority must be provided per Tx Protocol. origin: Standard AUTOSAR configuration parameter.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: dependency: DcmDslProtocolID		ECU

SWS Item	--		
Name	DcmDslProtocolIsParallelExecutable		
Description	Enables the parallel processing of ROE or Periodic Transmission protocol. Only these both protocols are allowed to run in parallel to normal protocol (UDS, OBD).		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	--		
Name	DcmDslProtocolTimeLimit {DCM_LIMITS}		
Description	Link to the extended / limit timing structure. Note: limits are checked when using AccessTimingParameter service. upperMultiplicity: Exactly one extended timing structure must be provided. lowerMultiplicity: Exactly one extended timing structure must be provided.		
Multiplicity	0..1		
Type	Reference to DcmDslProtocolTimingRow		
ConfigurationClass	Pre-compile time	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	--		
Name	DcmDslProtocolBufferID {DCM_PROTOCOL_RX_BUFFER_ID}		
Description	Link to buffer configuration (DcmDslBufferID) for configuration of protocol buffer. For all the Rx and Tx actions one buffer is shared for one protocol		

	upperMultiplicity / lowerMultiplicity:: Exactly one Rx/Tx buffer is required for protocol reception and transmission.		
Multiplicity	1		
Type	Reference to DcmDslBuffer		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope:		ECU
	dependency: DcmDslProtocolRxPduld, DcmDslBuffer.DcmDslBufferID		

SWS Item	--		
Name	DcmDslProtocolSIDTable {DCM_PROTOCOL_IDENTIFIER_TABLE_ID}		
Description	Link to the used diagnostic service table for this protocol. upperMultiplicity: Must have exactly one service table for the protocol. lowerMultiplicity: Must have exactly one service table for the protocol.		
Multiplicity	1		
Type	Reference to DcmDsdServiceTable		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope:		ECU
	dependency:		DcmDslProtocolID, DcmDsdServiceIdTable.DcmDsdSidTabId

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDslConnection	1..*	This container contains links between Diagnostic Protocol (=DcmDslProtocolRow) and the according Rx or Tx channel. Because of the usecase to allow more then one diagnostic tester (using different CAN channels) it is necessary to allow configuration of multiple DcmDslConnections.

10.4.8 DcmDslProtocolRx

SWS Item	--		
Container Name	DcmDslProtocolRx{DIAGNOSTIC_PROTOCOL_RX_TABLE}		
Description	This container contains the configuration (parameters) for the protocol configuration of RX channel (for each protocol) The following parameters needs to be configured per protocol. Please keep in mind, that the parameter DcmDslProtocolRxPduld can be given several times per protocol and that the combination from DcmDslProtocolRxPduld and DcmDslProtocolRxAddrType is unique. Only one physical protocol is allowed per connection. upperMultiplicity: More than one receive PDU ID may be configured for reception (e.g. one for func requests, one for phys requests). lowerMultiplicity: At least one receive PDU ID configuration must be provided per protocol.		
Configuration Parameters			

SWS Item	--		
Name	DcmDslProtocolRxAddrType {DCM_PROTOCOL_RX_ADDR_TYPE}		
Description	Declares the communication type of this DCM_PROTOCOL_DCMRXPDUID. PHYSICAL is used for 1 to 1 communication FUNCTIONAL is used for 1 to n communication		
Multiplicity	1		
Type	EnumerationParamDef		

Range	FunctionalType	FUNCTIONAL = 1 to n communication	
	PhysicalType	PHYSICAL = 1 to 1 communications using physical addressing	
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: dependency: DcmDslProtocol.DcmDslProtocolRow.DcmDslProtocolID		ECU DcmDslProtocolRxPdul,

SWS Item	--		
Name	DcmDslProtocolRxPduRef {DCM_PROTOCOL_DCMRXPDUID}		
Description	DcmRxPdul reference for reception of requests / Multiple links shall be allowed. (e.g. one DcmRxPdul to receive func requests, one DcmRxPdul to receive phys requests)		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: dependency: DcmDslProtocolRxAddrType, DcmDslProtocolRxPduRef		ECU

No Included Containers

10.4.9 DcmDslProtocolTiming

SWS Item	Dcm031 :		
Container Name	DcmDslProtocolTiming{PROTOCOL_TIMING_STRUCTURE}		
Description	This container contains the configuration (parameters) for Protocol timing. This container contains rows of DcmDslProtocolTimingRow containers. Hint: DcmDslProtocolTiming is only relevant in case of using the UDS service AccessTimingParameter.		
Configuration Parameters			

Included Containers

Container Name	Multiplicity	Scope / Dependency
DcmDslProtocolTimingRow	0..*	Definition of a single row of configuration for protocol timing

10.4.10 DcmDslProtocolTimingRow

SWS Item	Dcm031 :		
Container Name	DcmDslProtocolTimingRow{PROTOCOL_TIMING_STRUCTURE_ROW}		
Description	Definition of a single row of configuration for protocol timing		
Configuration Parameters			

SWS Item	Dcm031 :		
Name	DcmTimStrS3Server {DCM_TIMSTR_S3SERVER}		
Description	This is the value for S3Server in milliseconds. This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of		

	DCM.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	Dcm031 :		
Name	DcmTimStrP2ServerMax {DCM_TIMSTR_P2SERVER_MAX}		
Description	This is the value for P2ServerMax in milliseconds. This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	Dcm031 :		
Name	DcmTimStrP2ServerMin {DCM_TIMSTR_P2SERVER_MIN}		
Description	This is the value for P2ServerMin in milliseconds. This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	Dcm031 :		
Name	DcmTimStrP2StarServerMin {DCM_TIMSTR_P2STARSERVER_MIN}		
Description	This is the value for P2*ServerMin in milliseconds. This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	Dcm031 :		
Name	DcmTimStrP2StarServerMax {DCM_TIMSTR_P2STARSERVER_MAX}		
Description	This is the value for P2*ServerMax in milliseconds. This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.4.11 DcmDslProtocolTx

SWS Item	--		
Container Name	DcmDslProtocolTx{DIAGNOSTIC_PROTOCOL_TX_TABLE}		
Description	This container contains the configuration (parameters) for the protocol configuration of TX channel (for each protocol) The included parameters needs to be configured per protocol.		
Configuration Parameters			

SWS Item	--		
Name	DcmDslProtocolTxPduRef {DCM_PROTOCOL_DCMTXPDUID}		
Description	DcmRxpDuld reference for transmission of responses		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope:		ECU
	dependency: DcmDslProtocol.DcmDslProtocolRow.DcmDslProtocolID		

No Included Containers

10.4.12 DcmDslResponseOnEvent

SWS Item	Dcm069 :		
Container Name	DcmDslResponseOnEvent{RESPONSE_ON_EVENT_PARAMETERS}		
Description	This container contains the configuration (parameters) for ResponseOnEvent service. Hint: ROE service request is receipt via DcmDslMainConnection and that Event response is given via DcmDslResponseOnEvent		
Configuration Parameters			

SWS Item	Dcm069 :		
Name	DcmDslRoeTxPduRef {DCM_ROE_DCMTXPDUID}		

Description	Reference to the PDU for transmission of ROE response (only needed for ROE Transmission Type is TYPE2) upperMultiplicity: One PDU required if ROE Transmission Type is TYPE2. lowerMultiplicity: No PDU required if ROE Transmission Type is TYPE1.		
Multiplicity	0..1		
Type	Reference to Pdu		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: dependency: DcmDslROEEnabled, DcmDslROETransType		ECU

No Included Containers

10.5 DSP (Diagnostic Service Processor)

10.5.1

SWS Item	--
Container Name	DcmDsp
Description	These parameters apply to Diagnostic Service Processing. There will always be one set of these parameters per DCM.
Configuration Parameters	

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDspSession	1	This container contains the configuration (DSP parameter) session control configuration (per session control) This container contains Rows of DcmDspSessionRow
DcmDspReadDTC	1	This container contains the configuration (parameters) of the service Read DTC Information - 0x19 (per sub-function type) This container contains Rows of DcmDspReadDTCRow
DcmDspSecurity	1	This container contains the configuration (DSP parameter) for security level configuration (per security level) Description This container contains Rows of DcmDspSecurityRow

10.5.2 DcmDspReadDTC

SWS Item	Dcm074 :
Container Name	DcmDspReadDTC{DCM_READDTC_SUB_FUNCTION_TABLE}
Description	This container contains the configuration (parameters) of the service Read DTC Information - 0x19 (per sub-function type) This container contains Rows of DcmDspReadDTCRow
Configuration Parameters	

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDspReadDTCRow	0..*	This container contains the configuration (parameters) of the service Read DTC Information - 0x19 (per sub-function type)

10.5.3 DcmDspReadDTCRow

SWS Item	Dcm073 :
-----------------	-----------------

Container Name	DcmDspReadDTCRow{DCM_READDTC_SUB_FUNCTION_TABLE_ROW}
Description	This container contains the configuration (parameters) of the service Read DTC Information - 0x19 (per sub-function type)
Configuration Parameters	

SWS Item	Dcm073 :		
Name	DcmDspDTCTInfoSubFuncSupp {DCM_DTC_INFO_SUB_FUNCTION_SUPPORT}		
Description	Indicates whether the respective sub-function is supported. TRUE = sub-function supported. FALSE = sub-function not supported.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	Dcm073 :		
Name	DcmDspDTCTInfoSubFuncLevel {DCM_DTC_INFO_SUB_FUNCTION_LEVEL}		
Description	hex value of the sub-function.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	--		
Name	DcmDspDTCTInfoSecLevelRef {DCM_SIDTAB_SEC_LEVEL_ROW}		
Description	Link to the Security Access Levels needed for execution of the DcmDspReadDTCRow. Please refer to the ISO 14229-1, ISO 15031-5 and "Verification of the Security Access levels" (chapter 7.2.2.4). For DcmDspSecurityLevel please refer to chapter 10.5 DSP (Diagnostic Service Processing)Please note, that it shall be provided to configure several DcmDsdSidTabSecLevelRef.		
Multiplicity	1..*		
Type	Reference to DcmDspSecurityRow		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.5.4 DcmDspSecurity

SWS Item	Dcm073 :
Container Name	DcmDspSecurity{DCM_SEC_LEVEL_TABLE}
Description	This container contains the configuration (DSP parameter) for security level configuration (per security level) Description This container contains Rows of DcmDspSecurityRow
Configuration Parameters	

Included Containers		
Container Name	Multiplicity	Scope / Dependency
DcmDspSecurityRow	0..31	Definition of a single Row of configuration for security level onfiguration (per security level)

10.5.5 DcmDspSecurityRow

SWS Item	Dcm073 :
Container Name	DcmDspSecurityRow{DCM_SEC_LEVEL_TABLE_ROW}
Description	Definition of a single Row of configuration for security level onfiguration (per security level)
Configuration Parameters	

SWS Item	Dcm073 :		
Name	DcmDspSecuritySeedSize {DCM_SEC_NUM_SEED}		
Description	size of the security seed (in Bytes).		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	--		
Name	DcmDspSecurityDelayTimeOnBoot		
Description	Start delay timer on power on (in ms) The AUTOSAR configuration standard is to use SI units, so this parameter is defined as float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM. min: A negative value is not allowed. upperMultiplicity: Exactly one delay time must be specified per security level configuration. lowerMultiplicity: Exactly one delay time must be specified per security level configuration.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	Dcm073 :		
Name	DcmDspSecurityLevel {DCM_SEC_LEVEL, DCM_SEC_NAME}		
Description	hex value of Security level 0 is the highest security level. 0x00, 0x01,0x03...0x7F: configuration dependent - Conversion formula to calculate SecurityLevel out of tester requested SecurityAccessType parameter: SecurityLevel = (SecurityAccessType + 1) / 2 Type: Dcm_SecLevelType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	Dcm073 :		
Name	DcmDspSecurityNumAttDelay {DCM_SEC_NUM_MAX_ATT_DELAY}		
Description	Number of failed security accesses after which the delay time is activated		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module dependency: DcmDspSecurityNumAttLock, DcmDspSecurityDelayTime		

SWS Item	Dcm073 :		
Name	DcmDspSecurityDelayTime {DCM_SEC_DELAY_INV_KEY}		
Description	Delay time after failed security access (in ms). This is started after DcmDspSecurityNumAttDelay number of failed security accesses. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM. min: A negative value is not allowed. upperMultiplicity: Exactly one delay time must be specified per security level configuration. lowerMultiplicity: Exactly one delay time must be specified per security level configuration. origin: Standard AUTOSAR configuration parameter.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	Dcm073 :		
Name	DcmDspSecurityNumAttLock {DCM_SEC_NUM_MAX_ATT_LOCK}		

Description	Number of failed security accesses after which security access is locked. 0 indicates that the security access is disabled and never locks.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: dependency: DcmDspSecurityNumAttDelay		Module

SWS Item	Dcm073 :		
Name	DcmDspSecurityKeySize {DCM_SEC_NUM_KEY}		
Description	size of the security key (in Bytes).		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.5.6 DcmDspSession

SWS Item	Dcm072 :		
Container Name	DcmDspSession{DCM_SESSION_LEVEL_TABLE}		
Description	This container contains the configuration (DSP parameter) session control configuration (per session control) This container contains Rows of DcmDspSessionRow		
Configuration Parameters			

Included Containers			
Container Name	Multiplicity	Scope / Dependency	
DcmDspSessionRow	0..31	Definition of a single Row of session control configuration (per session control)	

10.5.7 DcmDspSessionRow

SWS Item	Dcm072 :		
Container Name	DcmDspSessionRow{DCM_SESSION_LEVEL_TABLE_ROW}		
Description	Definition of a single Row of session control configuration (per session control)		
Configuration Parameters			

SWS Item	Dcm072 :		
-----------------	-----------------	--	--

Name	DcmDspSessionP2ServerMax {DCM_SESSION_P2SERVER_MAX}		
Description	This is the session value for P2ServerMax in milliseconds (per Session Control). This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	Dcm072 :		
Name	DcmDspSessionP2StarServerMax {DCM_SESSION_P2STRSERVER_MAX}		
Description	This is the session value for P2*ServerMax in milliseconds (per Session Control). This is defined in the AUTOSAR SWS for DCM as a uint16. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	Dcm072 :		
Name	DcmDspSessionLevel {DCM_SESSION_LEVEL, DCM_SESSION_NAME}		
Description	hex value of the Session control. The symbolicName represents the Name of the Session SWS defines the Dcm_SesCtrlType: 0x01 DEFAULT_SESSION 0x02 PROGRAMMING_SESSION 0x03 EXTENDED_DIAGNOSTIC_SESSION 0x04 SAFETY_SYSTEM_DIAGNOSTIC_SESSION 0x05...0x7F <configuration dependent (according "diagnosticSessionType" parameter DiagnosticSessionControl request> 0x80...0xFE Reserved by Document 0xFF ALL_SESSION_LEVEL		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.5.8 DcmGeneral

SWS Item	DCM075, DCM076 :		
Container Name	DcmGeneral{COMPONENT_WIDE_PARAMETERS}		
Description	This container contains the configuration (parameters) for Component wide parameters		
Configuration Parameters			

SWS Item	DCM075, DCM076 :		
Name	DcmVersionInfoApi {DCM_VERSION_INFO_API}		
Description	Preprocessor switch to enable or disable the output Version info of the functionality.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	DCM075, DCM076 :		
Name	DcmRequestIndicationEnabled {DCM_REQUEST_INDICATION_ENABLED}		
Description	Allows to enable or disable the requested indication mechanism.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	DCM075, DCM076 :		
Name	DcmDevErrorDetect {DCM_DEV_ERROR_DETECT}		
Description	Preprocessor switch to enable or disable the Development Error Detection (DET) mechanism.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	DCM075, DCM076 :		
Name	DcmTaskTime {DCM_TASK_TIME}		
Description	Allow to configure the time for the periodic cyclic task (in ms). Please note: This configuration value shall be equal to the value in the SchedulerManager module. The AUTOSAR configuration standard is to use SI units, so this parameter is defined as float value in seconds. DCM configuration tools must convert this float value to the appropriate value format for the use in the software implementation of DCM. min: A negative value is not allowed. upperMultiplicity: Exactly one TaskTime must be		

	specified per configuration. lowerMultiplicity: Exactly one TaskTime must be specified per configuration.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.5.9 DcmDslDiagResp

SWS Item	--		
Container Name	DcmDslDiagResp{DIAGNOSTIC_RESPPEND_CFG}		
Description	This container contains the configuration (parameters) for the ResponsePending handling		
Configuration Parameters			

SWS Item	--		
Name	DcmDslDiagRespMaxNumRespPend {DCM_NUM_MAX_RESPPEND}		
Description	Maximum number of negative responses with response code 0x78 (requestCorrectlyReceived-ResponsePending) allowed per request DCM will send a negative response with response code 0x10 (generalReject), in case the limit value gets reached.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	--		
Name	DcmDslDiagRespForceRespPendEn		
Description	Allow to enable (TRUE) or disable (FALSE) the mechanism of directly triggering of ResponsePending by application.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

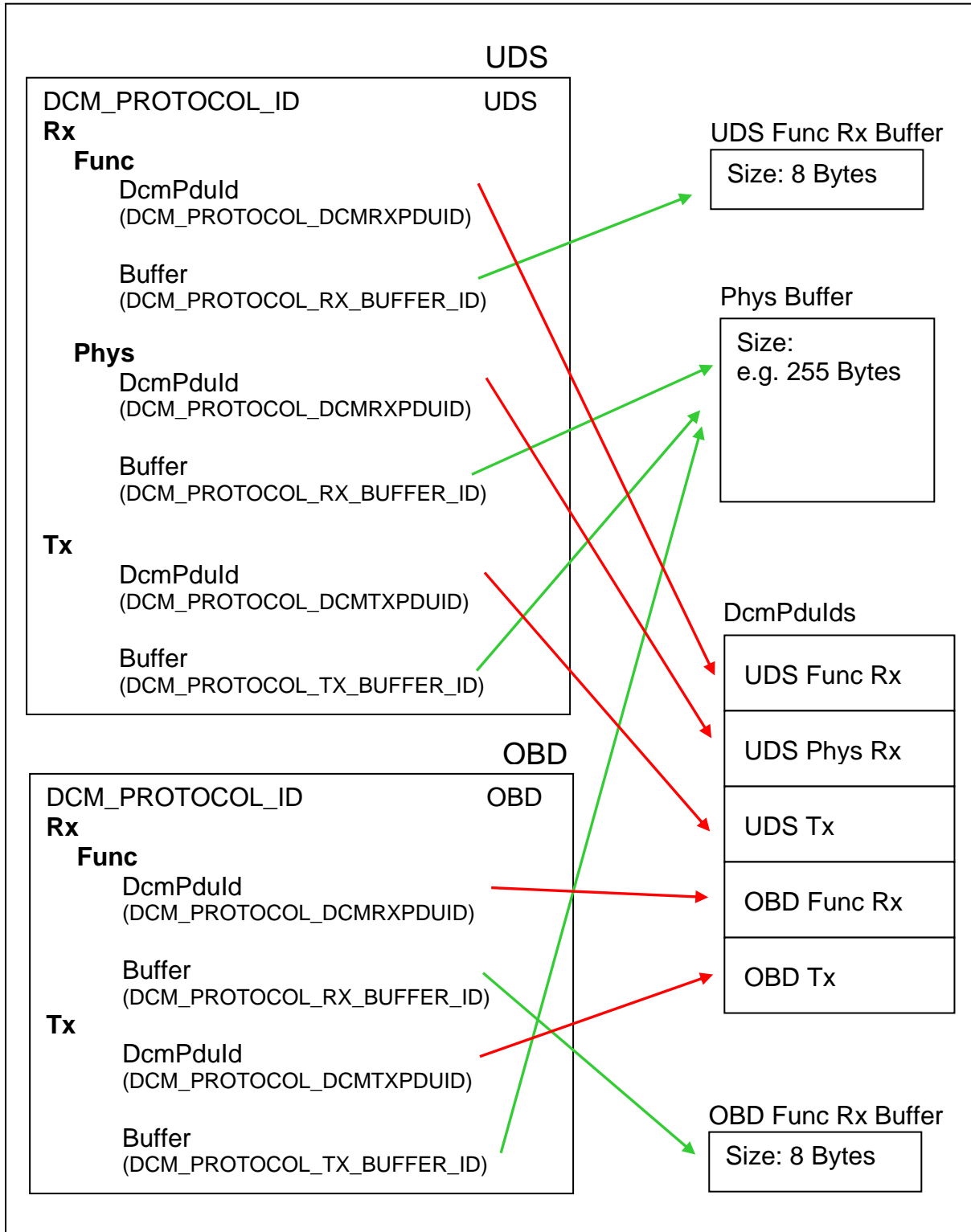


Figure 16 Examples of protocol configuration with focus on buffer / DcmPduId settings

Above example shows protocol configuration at the use cases examples OBD and UDS (used for customer enhanced diagnosis).

It is assumed that for UDS communication, there are functional and physical requests. There will be separate DcmPduRxDs for functional and physical reception.

Concerning buffer configuration it is proposed to use a separate buffer for the functional requests. This in correspondence to support the keep alive logic with functional addressed TesterPresent commands (see chapter 7.2.1.4.2.2 Concurrent “TesterPresent” (“keep alive logic”)).

It is also proposed to use a separate receive buffer for the OBD commands. This in reference to support the protocol switch functionality (see chapter 9.2.6 Process concurrent requests).

It is allowed to share for both protocols the transmit buffer.

In the following Figure the dependency between the DcmDslMainConnection and DcmDslProtocolRx, DcmDslProtocolTx and DcmDslProtocol is shown.

Please note:

The DcmDslProtocolRx has two possible configurations:

- functional
- physical

The physical shall have a 1:1 (or 1:0) dependency to the DcmDslMainConnection. (which means: DcmDslProtocolRxPduRef in combination DCM_PROTOCOL_RX_ADDR_TYP = physical can exist only once per “Module”)

The functional shall have a 1:n dependency to the DcmDslMainConnection. (which means: DcmDslProtocolRxPduRef in combination DCM_PROTOCOL_RX_ADDR_TYP = functional can exist several times per “Module”)

The DcmDslProtocolTx shall exist only once per “Module”

10.6 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_AR_MINOR_VERSION),
arPatchVersion (<Module>_AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_SW_MINOR_VERSION),
swPatchVersion (<Module>_SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [8] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

11 Changes to Release 1

11.1 Deleted SWS Items

<i>SWS Item</i>	<i>Rationale</i>
Not traced	Not traced

11.2 Replaced SWS Items

<i>SWS Item of Release 1</i>	<i>replaced by SWS Item</i>	<i>Rationale</i>
Not traced	Not traced	Not traced

11.3 Changed SWS Items

<i>SWS Item</i>	<i>Rationale</i>
Not traced	Not traced

11.4 Added SWS Items

<i>SWS Item</i>	<i>Rationale</i>
DCM047	Error types definition
DCM048	Api parameter check
DCM049	Development error notification
DCM050	Definition of Development Error Tracer
DCM051	Api definition for Development Error Tracer
DCM052	Module Id definition