Document Title	Specification of Diagnostic Communication Manager
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.1.1
Document Status	Final
Part of Release	2.1
Revision	0014

Document Change History

Date	Version	Changed by	Change Description
24.01.2007	2.1.1	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
05.12.2006	2.1.0	AUTOSAR Administration	• 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. • 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.

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

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.

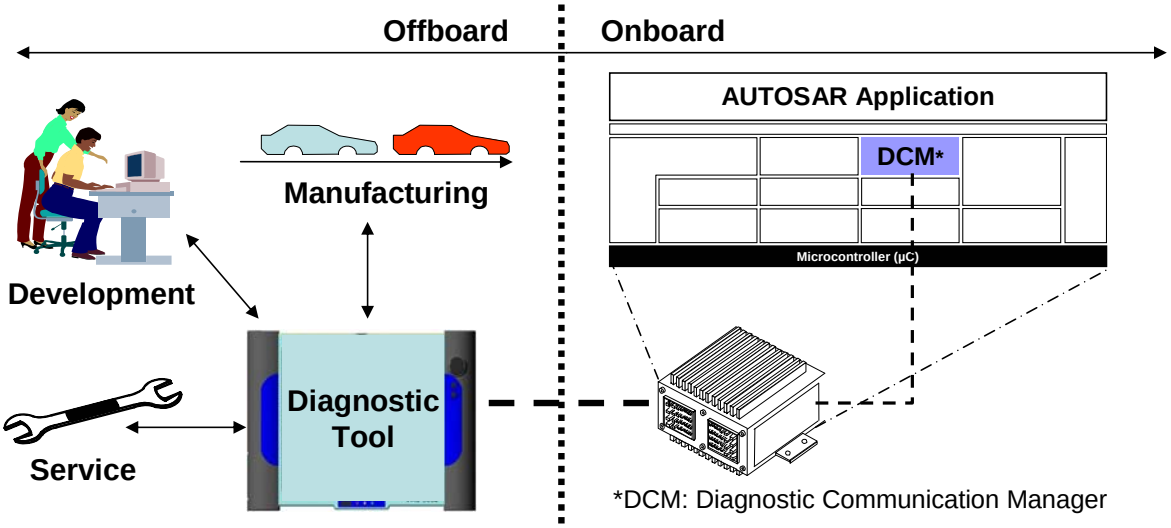


Figure 1 Communication overview between external diagnostic tools and AUTOSAR-SW

Dcm018: Dcm019: The DCM handles different diagnostic protocols, for legislated OBD (ISO 15031-5) and 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

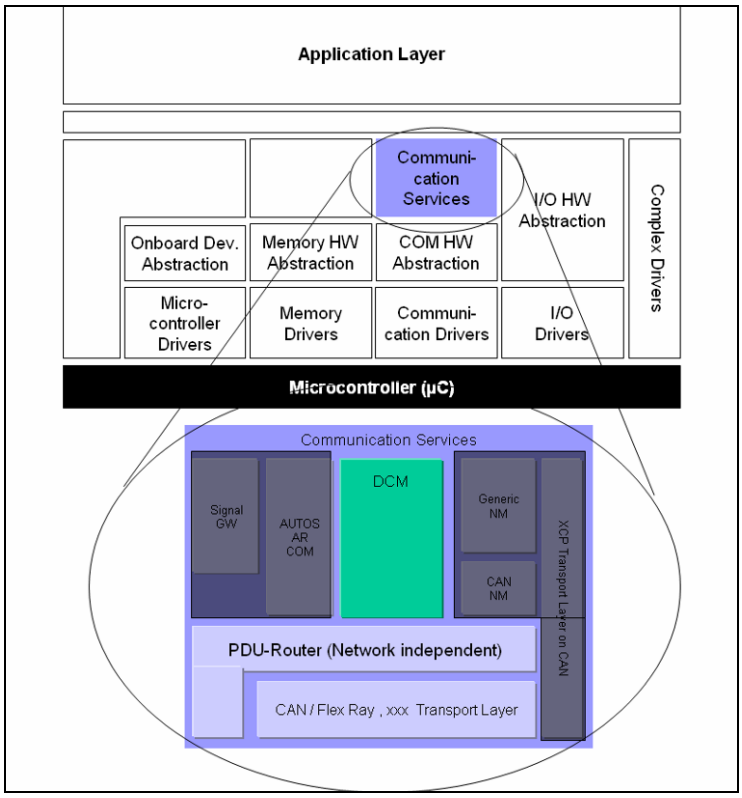


Figure 2 Position of DCM in AUTOSAR Architecture

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	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. • development • manufacturing • service (garage) Know devices are e.g. • a diagnostic tester • an OBD scan tool The external diagnostic tool is to be connected by a mechanic to gather information from “inside” the car.
Freeze Frame	A set of the vehicle/system operation conditions at a specific time.

Functional addressing	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). Use cases are (for example) broadcasting messages as “ECUReset” or “CommunicationControl” OBD communication will always be done in Functional addressing mode.
internal diagnostic tool	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. • advanced event tracking • advanced analysis’s for Service. The behavior of the device/ECU could be the same as if it was an external diagnostic tool. The meaning of ´internal diagnostic tool´ is NOT that it is included in each ECU as a AUTOSAR SW-Component.
Physical addressing	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). Use cases are (for example) messages as “ReadDataByIdentifier” or “InputOutputControlByIdentifier”

3 Related documentation

3.1 Input documents

[1] AUTOSAR General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[2] Specification of Module PDU Router
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_PDU_Router.pdf

[3] AUTOSAR Requirements on Basic Software Module Diagnostic
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_Diagnostic.pdf

[4] Specification of Module Communication Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_ComManager.pdf

[5] AUTOSAR Layered Software Architecture
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[6] AUTOSAR ECU Configuration Specification
https://svn.autosar.org/repos/10Releases/
AUTOSAR_ECU_Configuration.pdf

[7] Specification of Diagnostics Event Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DEM.pdf

3.2 Related standards and norms

[8] D1.5-General Architecture; ITEA/EAST-EEA, Version 1.0; chapter 3, page 72 et seq.

[9] D2.1-Embedded Basic Software Structure Requirements; ITEA/EAST-EEA, Version 1.0 or higher

[10] D2.2-Description of existing solutions; ITEA/EAST-EEA, Version 1.0 or higher.

[11] 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”]

[12] ISO15031-5.4 Communication between vehicle and external equipment for emmissions-related diagnostics - Part 5: Emissions-related diagnostic services (2005-01-13)

[13] ISO15765-3: Diagnostics on controller area network (CAN) - Part 3: Implementation of unified diagnostic services (UDS on CAN) (2004-10-06)

[14] 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

Dcm014: 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 only an API based on the diagnostic Service Request Identifier. That means the implementation described within this document provides only a partial analysis of the received diagnostic request data stream. It is up to the application to analyze and process the respective request data stream excluding the diagnostic request service identifier.
  However, within the functional block called DSP (see chapter 7.2.3) the DCM processes already a sub-set of important diagnostic service requests (e.g. fault memory access, session control). Future implementations should extend the DSP functionality to embed the analysis of the diagnostic data stream completely within DCM (for as much services as possible).

- 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)

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

Dcm056: 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**

As mentioned before the DCM provides an API, based on the diagnostic Service Request Identifier, with the capability of a partial analysis of the received diagnostic request data stream. All functionalities related to diagnostic communication, as protocol handling, timing and segmented responses (paged buffer) shall be done by the DCM. Furthermore the DCM shall handle a subset of important diagnostic services (like ClearDiagnosticInformation or ReadDTCInformation). Current implementations of the different diagnostic services vary for each of the OEMs, so that from the DCM point of view, the application SW-components shall implement the functional part of the requested services.
To address these application SW-components, a Central Diagnostic SW-component shall be set up above the RTE in the application software. The Central Diagnostic SW-component shall execute several Diagnostic tasks, which are mostly high OEM specific.
The RTE shall connect the DCM and the Central Diagnostic SW-component by an 1:1 mapping.
The Central Diagnostic SW-component is the responsibility of the developer of the application. The main Task of the Central Diagnostic SW-component will be the receiving of a diagnostic message (via the/a function call), to preprocess or analyze the message and to distribute / discharge the task by calling the functionalities of other Autosar Software components or other Autosar Base modules (e.g. NVRAM Manager).
This proceeding describes very well the solutions which are implemented right now in not Autosar ECU’s.

The interaction between the Central Diagnostic SW-component and the application SW-components, which handles the functional part of a diagnostic request, is **NOT** given within this specification.

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

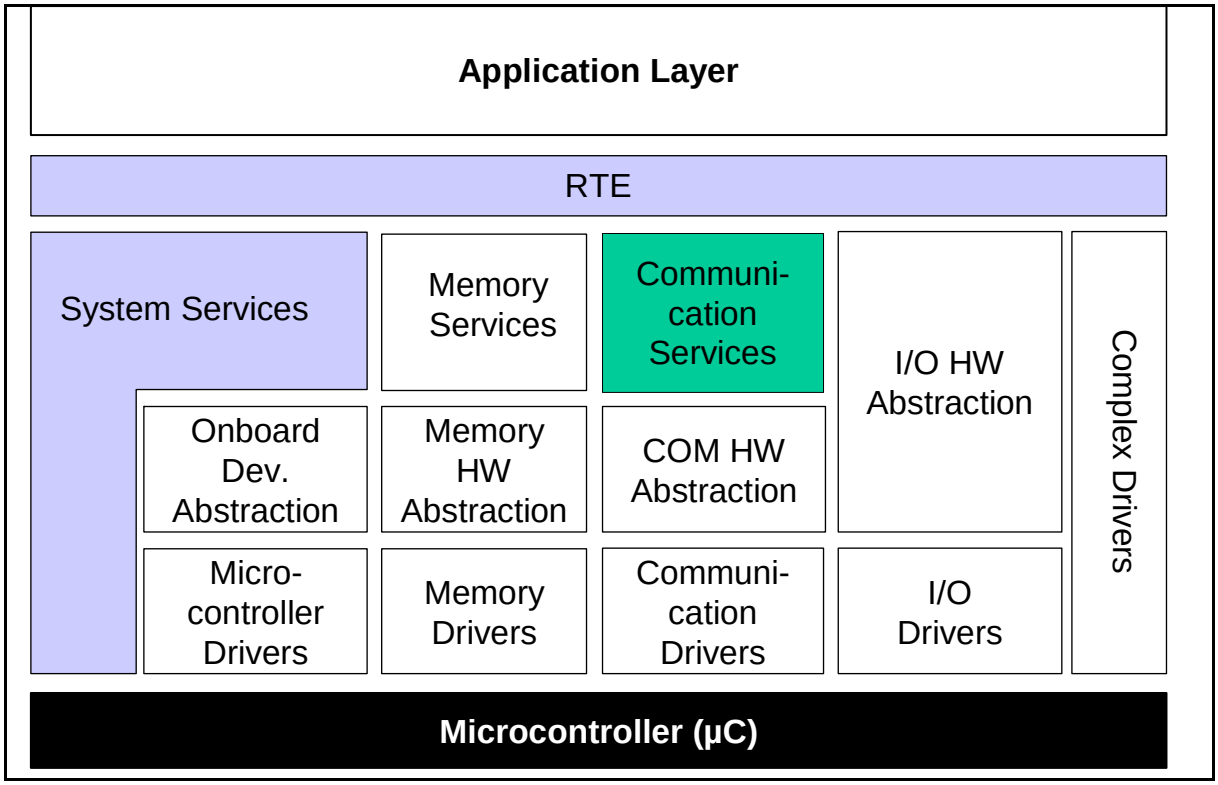


Figure 3: Overview of the DCM interfaces

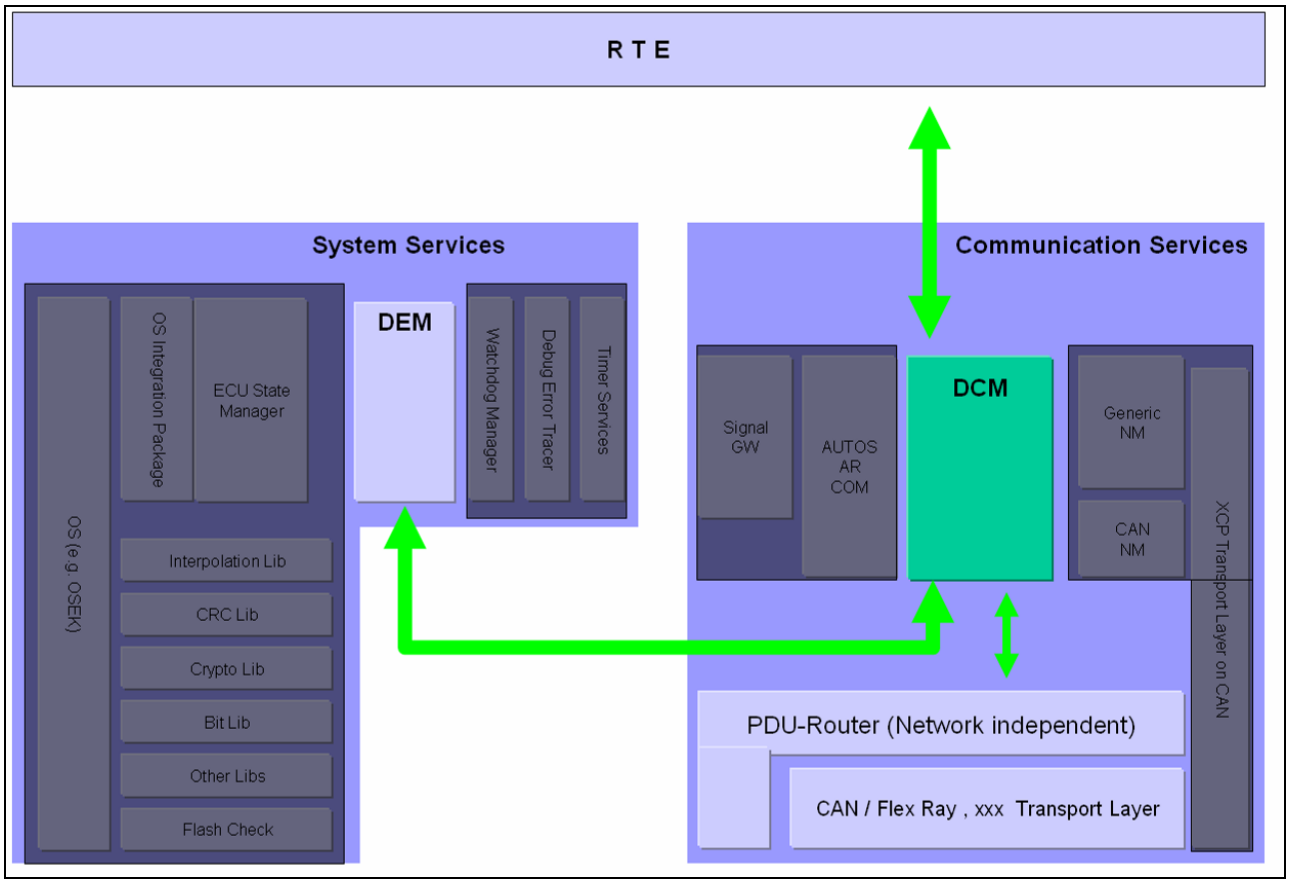


Figure 4: DCM interfaces

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.
Dcm066:
Files holding configuration data for AUTOSAR Basic SW modules shall have a format that is readable and understandable by human beings

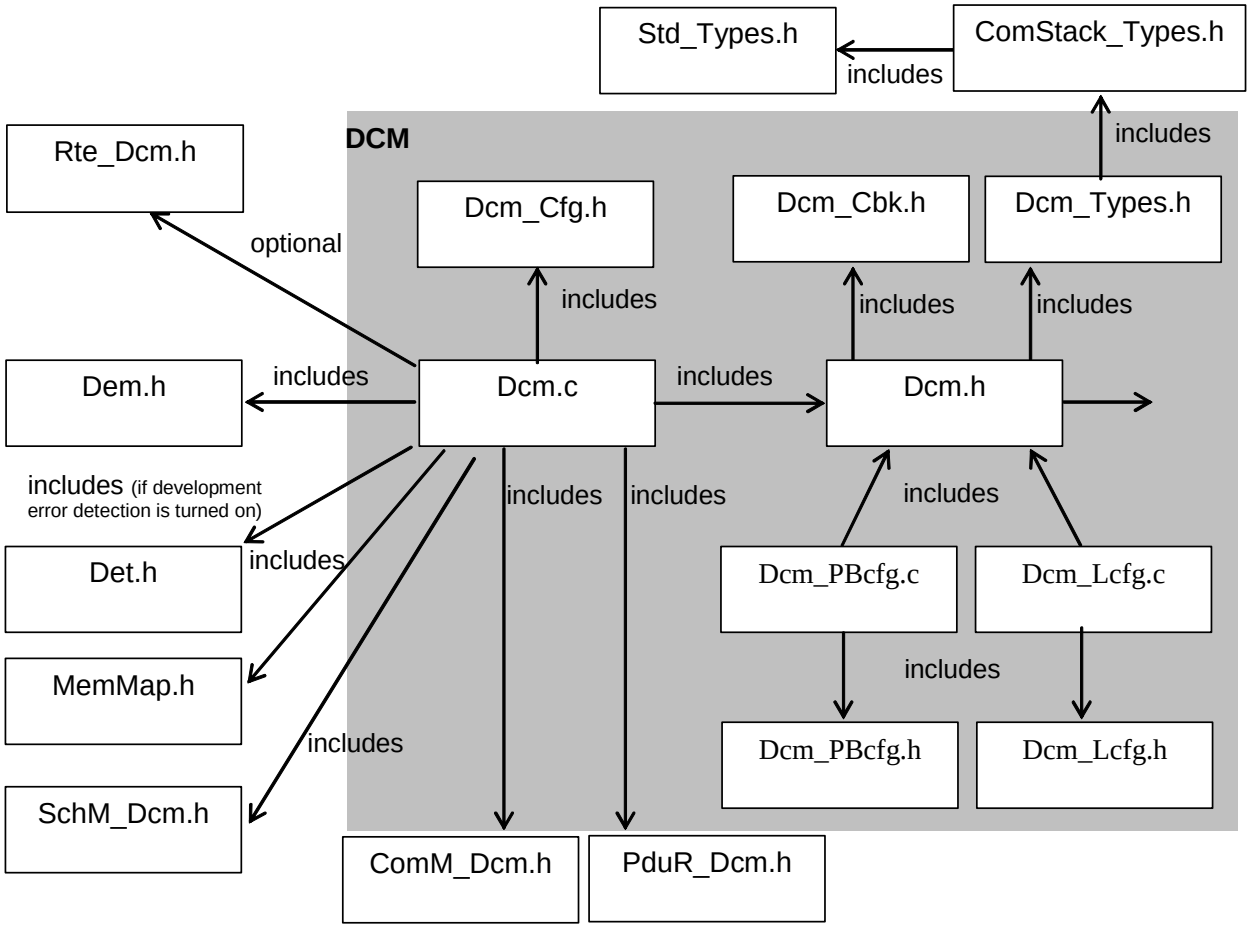


Figure 5: DCM Component file structure

5.1.2 Header file structure

Dcm055:

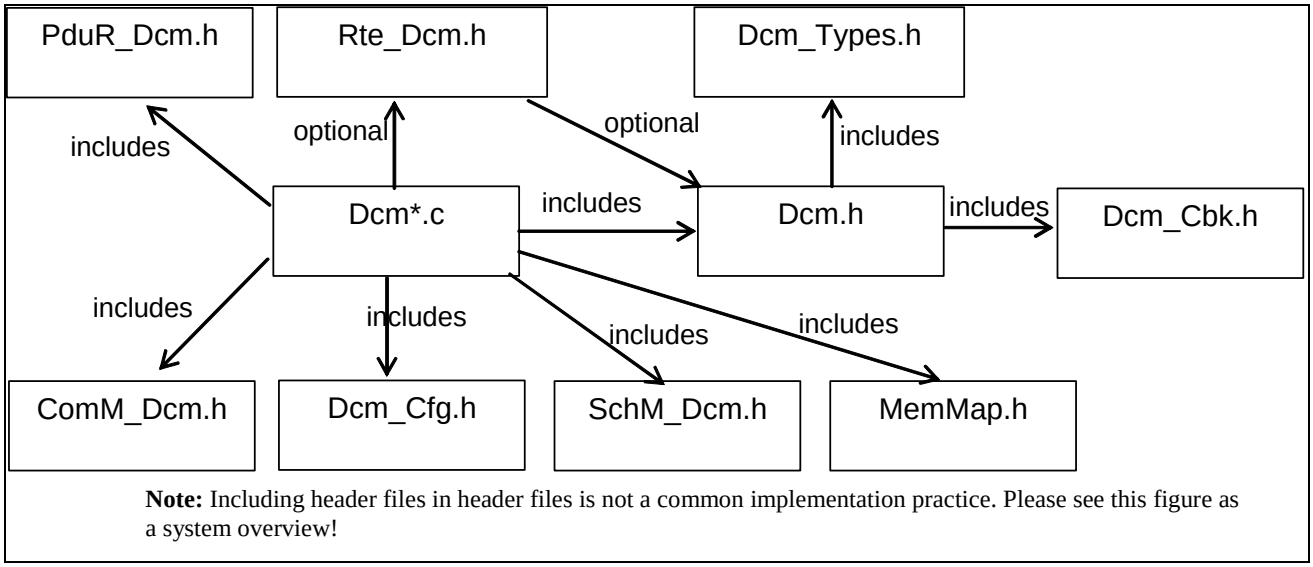


Figure 6: DCM Component header file structure

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.h` shall include `Dcm_Types.h`
- `Dcm.c` shall include `Dcm.h` and `SchM_Dcm.h`
- `Dcm.h` shall include `Dcm_Cfg.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	Dcm056
[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	Dcm066
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	Dcm014
[BSW04080] Support multi-channel capability for diagnostic communication	Dcm014
[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	Dcm014
[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 and “Central Diagnostic SW Component”).
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 application (AUTOSAR SW-Component or 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

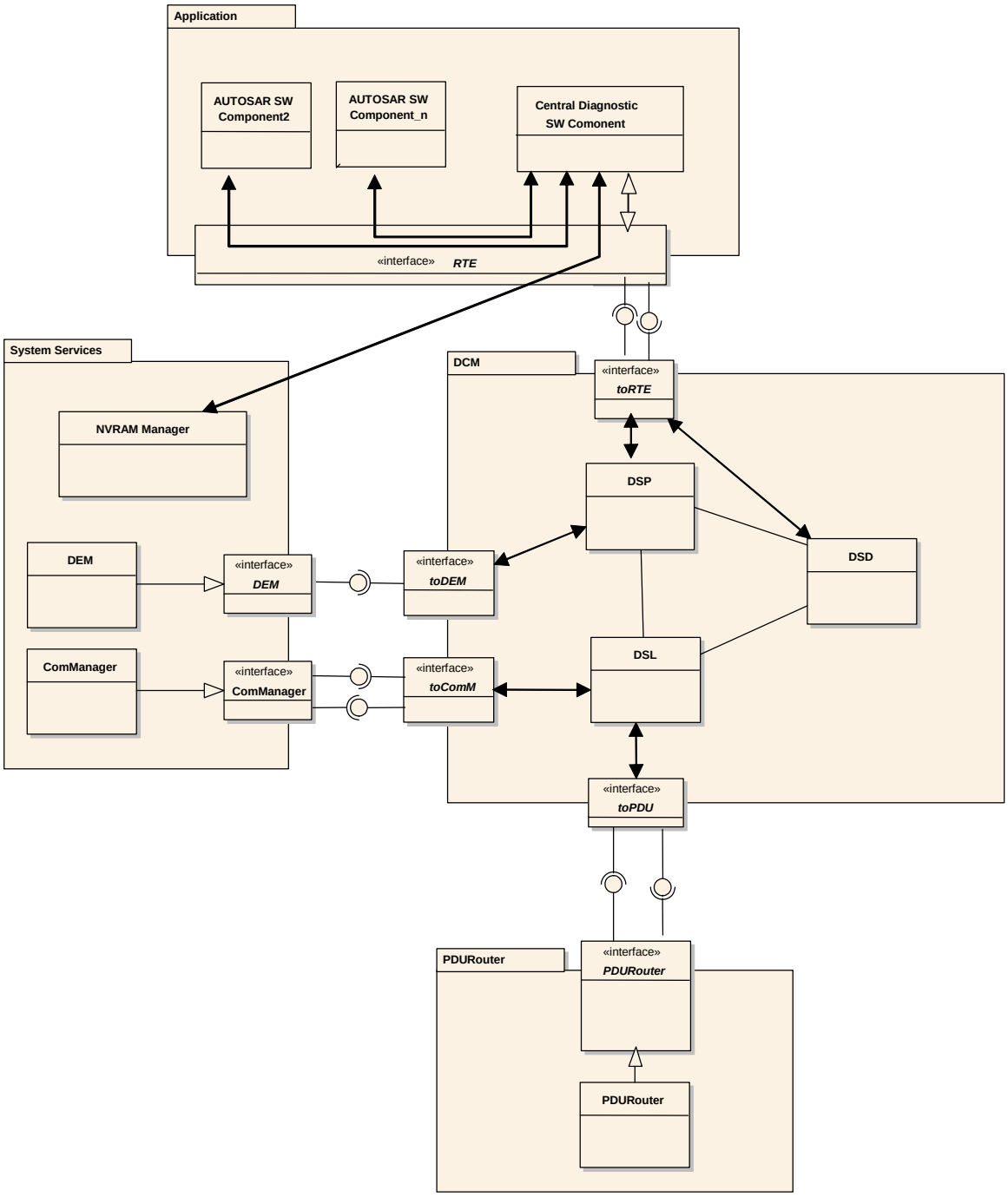


Figure 7: DCM Architecture overview with interfaces

Note: The interface between the Central Diagnostic Component and other AUTOSAR BSW or SW Components is not in focus of this specification and shall be seen as a hint.

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 10.3 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**
  - PDU Router provides data of incoming diagnostic requests
  - DSL triggers output of diagnostic responses
- **DSD** (Diagnostic Service Dispatcher)
  - DSL informs DSD about incoming requests and provides the data
  - DSD triggers output of diagnostic responses
- **Software Components / DSP** (Diagnostic Service Processing)
  - DSL provides access to security and session state
- **Communication Manager**
  - 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 (“PagedBuffer”)

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 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 DcmRxPduId, 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.

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 DcmPduIds for different diagnostic communication application.

Example:

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

Address Type (physical / functional addressing) is configured per DcmRxPduId (see chapter 10.3 DSL (Diagnostic Session Layer)). A configuration per DcmRxPduId is possible, since there will be always different DcmRxPduId 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 DcmRxPduId (UDS func DcmRxPduId) 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 10.3 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. Ongoing response handling in application (in case of PagedBuffer processing) needs to be stopped (**RTE_DcmCancelPagedBufferProcessing()**).

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).

Sequence diagram is available in chapter 9.2.2 Process Busy behavior

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

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

According to the communication mode (see chapter 7.2.1.4.2.15 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 ReadDataByPeriodicIdentifie

Referenced configuration parameters can be found in chapter 10.3 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 application 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 ***DCM_ROE_TRANS_TYPE***.
In case of TYPE2, the separate channel shall be configured with ***DCM_ROE_DCMTXPDUID***

Responses generated by “ResponseOnEvent” use a separate buffer of configurable size(configuration parameter DCM_ROE_TX_BUFFER_ID). The configured buffer can be used for transmission and reception of the ROE messages. The content of the ***pMsgContext*** pointer (ROE message) shall be copied into the buffer.

According to the communication mode (see chapter 7.2.1.4.2.15 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 10.3 DSL (Diagnostic Session Layer).

7.2.1.4.2.7 Support of segmented response (“PagedBuffer”)

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

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 9.3.6 Process Service Request with PagedBuffer.

Please note: “PagedBuffer” handling is for transmit only / no support for reception.

7.2.1.4.2.8 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, whenever session is changed from

a session other then defaultSession [including the currently active diagnostic session] (**Dcm_SetSecurityLevel()**) (initiated by service 'DiagnosticSessionControl' or $S3_{Server}$ timeout)
More explanation can be found in [6] 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).

7.2.1.4.2.9 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.

7.2.1.4.2.10 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.

7.2.1.4.2.11 Informs depending modules on session change

Whenever the value of the active session changes (initiated by service "DiagnosticSessionControl" or $S3_{Server}$ timeout), DSL informs the application by activation of the following function: **RTE_DcmSesCtrlChangeIndication()**.

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)

7.2.1.4.2.12 Allow to modify timings

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

$P2_{ServerMin}$, $P2_{ServerMax}$, $P2^*_{ServerMin}$, $P2^*_{ServerMax}$, $S3_{Server}$
Definition of these parameters can be found in the ISO15765-3 document.

Each protocol (OBD, enhanced diagnosis) can have its own protocol timing values (given by protocol configuration). These timing values are set, when the protocol is started (More description to "protocol start" is given in chapter 7.2.1.4.2.13 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 ***DCM_PROTOCOL_TIME_LIMIT*** *(*see chapter 10.3 DSL (Diagnostic Session Layer))
- **Dcm_SetSesTimingValues():** to set the new timing parameters. Since activation of new timing values is only allowed after sending the response, application needs to call this function in the response confirmation function: **RTE_DcmConfirmation()**

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

7.2.1.4.2.13 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 (e.g protocol timing parameter,..)
- different valid service table
- prioritization / preemption of protocol

Different protocol setting

The different diagnostic protocols may have different protocol settings, as e.g. protocol timing parameter ($P2_{ServerMin}$,$P2_{ServerMax}$,...)
Configuration details can be found in chapter 10.3 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 ***DCM_PROTOCOL_IDENTIFIER_TABLE***) referenced in the protocol configuration (see chapter 10.3 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 then the enhanced diagnosis. This can be supported with giving the protocol a priority information (see configuration parameter ***DCM_PROTOCOL_PRIO***) 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 DcmRxPduId values (configured per protocol, see configuration parameter ***DCM_PROTOCOL_DCMRXPDUID***) 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 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 by calling **RTE_DcmStopProtocol()** and finish with Dcm_ProcessingDone(). For further details take a look at the sequence charts 9.2.6 Process concurrent requests.

Detection of protocol start
Dcm036: With first request of a diagnostic protocol (OBD, enhanced diagnosis,..), DSL informs the application with following checkcondition callback function: **RTE_DcmStartProtocol()** (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 (see configuration structure ***DCM_PROTOCOL_TIME_DEFAULT*** referenced in the protocol configuration).
- Service table is set (see configuration parameter ***DCM_PROTOCOL_IDENTIFIER_TABLE*** referenced in the protocol configuration).

Also the security state and the session state are reset.

7.2.1.4.2.14 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”))

7.2.1.4.2.15 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 by the application.

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 by the application.

Full Communication:
With the callback routine Dcm_ComM_FullComModeEntered(), called by the Communication Manager, all kind of communication shall be disabled. 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 9.3.4)

Session Transitions:
If the actual diagnostic session shall be changed with the function **RTE_DcmSesCtrlChangeIndication()** 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 **RTE_DcmSesCtrlChangeIndication()** or with the elapse of the timer “$S3_{Server}$” 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 then 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 then 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 (e.g. Paged buffer)

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

If the application 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 application defers some processing ("asynchronous"). The response pending mechanism is covered by the DSL. Again, the application triggers the transmission explicitly, but from within the application (AUTOSAR SW-Component or DSP) context.

As soon as a request message is received, the corresponding DcmPduId 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 DcmPduId 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. This means that the transmission of a positive response is suppressed, but the transmission post-processing is simulated: The application will receive a confirmation for the successful transmission.

This means that it does not affect the application whether the response message is to be sent or not.
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 DcmTxPduId already used for normal diagnostic responses (single frames only); Type 2 = UUDT message on a separate DcmTxPduId. 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 DcmTxPduId already used for normal diagnostic responses, Way 2 = USDT messages on separate DcmTxPduId.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 functionalityDSL 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)

Direction	***Explanation***
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 a Diagnostic Message 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 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.

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 being (previously) determined diagnostic service identifier.

In this check the being (previously) determined 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 being (previously) determined diagnostic service identifier. The diagnostic service identifier is not supported when it is not included in the “Service Identifier Table“.
If the being (previously) determined diagnostic service identifier is supported, the data content of the Diagnostic Message will be stored in the DCM parameter Dcm_MsgContextType.
(For a definition of the Dcm_MsgContextType please refer to chapter 8.3.1.8 Dcm_MsgContextType)

If the (previously) determined diagnostic service identifier is not supported, the negative Response “Service not supported” (NRC 0x11) will be transmitted to the DSL.
The (previously) determined 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 Application (AUTOSAR SW Component or DSP).

7.2.2.4.2 Handling of „suppressPosRspMsgIndicationBit“

Dcm017: The suppressPosRspMsgIndicationBitis 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 parameterDcm_MsgContextType (Element: Dcm_MsgAddInfo).

In case of responsePending: suppressPosRspMsgIndicationBit should 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 „DCM_SIDTAB_SUBFUNC_AVAIL” configuration is set for the being (previously) determined 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 8.4.2.1 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 being (previously) determined diagnostic service identifier.
Therefore for the “DCM_SIDTAB_SESSION_LEVEL” configuration shall be searched in the “Service Identifier Table” (the being (previously) determined diagnostic service identifier).

When the being (previously) determined diagnostic service is allowed within the current Diagnostic Session the “Distribution of Diagnostic Message to data processor**”** can be started. If the being (previously) determined 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 being (previously) determined diagnostic service identifier.
Therefore for the “DCM_SIDTAB_SEC_LEVEL” configuration shall be searched in the “Service Identifier Table” (for the being (previously) determined diagnostic service identifier).

When the being (previously) determined diagnostic service is allowed with the current Security level the “Distribution of Diagnostic Message to data processor**”** can be started.
If the being (previously) determined 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 DCM_REQUEST_INDICATION_ENABLED set to TRUE, see chapter 10.2 DCM global Configurations), DSD calls the function RTE_DcmIndication() 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) in the “Service Identifier Table” for the being (previously) determined diagnostic service identifier, to call this Application function and provide the Diagnostic Message (in the format of Dcm_MsgContextType) to this Application function. For a definition of the Dcm_MsgContextType please refer to chapter 8.3.1.8 Dcm_MsgContextType)

Search for the executable functionality in the “Service Identifier Table”:
The “Service Identifier Table” includes an entry for the “DCM_SIDTAB_FUNCTIONPOINTER” element which determines the function pointer to the function of the DSP or the Application (basic SW modules via the standardized RTE Interface).

This entry is configurable. Names for the “DCM_SIDTAB_FUNCTIONPOINTER” 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.
In these functions of the application the implementation of the requested service shall be realized costumer specific by the developer of the AUTOSAR SW Component or the DSP.

Provision of the Diagnostic Message:
The application function shall be called with the parameter “Dcm_MsgContextType” which includes the Diagnostic Message.

In the example “WriteDataByIdentifier” the function XXX_DcmWriteDataByIdentifier(Dcm_MsgContextType) will be called. In this function the realizing of the “Write Data” shall be implemented customer specific by the developer of the AUTOSAR SW Component or the DSP.

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)

The finished execution of the requested Diagnostic Service is indicated in any case by the “Dcm_ProcessingDone” functionality executed by the Application (AUTOSAR SW Component or DSP).

Positive Response:
The positive Result of the Application (or 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 Application (or 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 0x31occurs

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 RTE_DcmConfirmation (Dcm_MsgContextType) to the Application (or 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 application (AUTOSAR SW Component or DSP) is waiting on the execution of the RTE_DcmConfirmation (Dcm_MsgContextType) functionality. So the DSD has to send this indication to the Application.

The processing of one Diagnostic Message of the Diagnostic Service Dispatcher (DSD) is finished with the "RTE_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 “RTE_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 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)

Note: the legislated emission related diagnostic service for reading Current Diagnostic Data data (service 0x01) (Emissions-related applications only) shall be handled in the application.

Unified Diagnostic Services:

Error memory 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)

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

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
- 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 [11] and the service specific configuration (for details pls. refer to 10.5).
  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 [11], 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.

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 is to be being (previously) determined 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

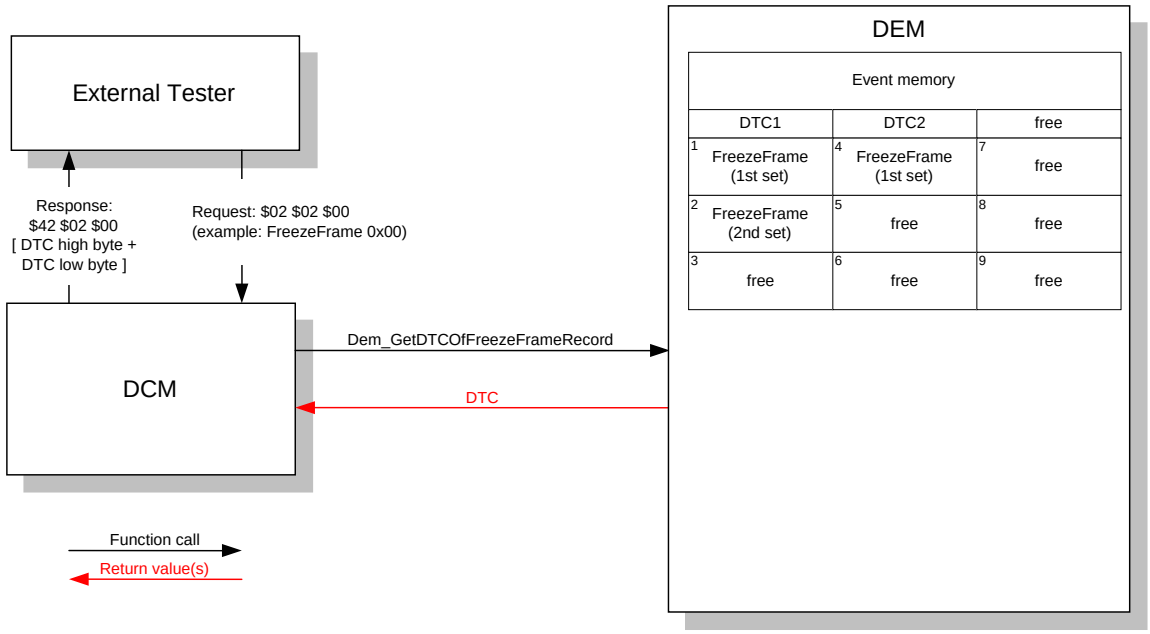


Figure 8: Request powertrain freeze frame data – Request DTC of Freeze Frame

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

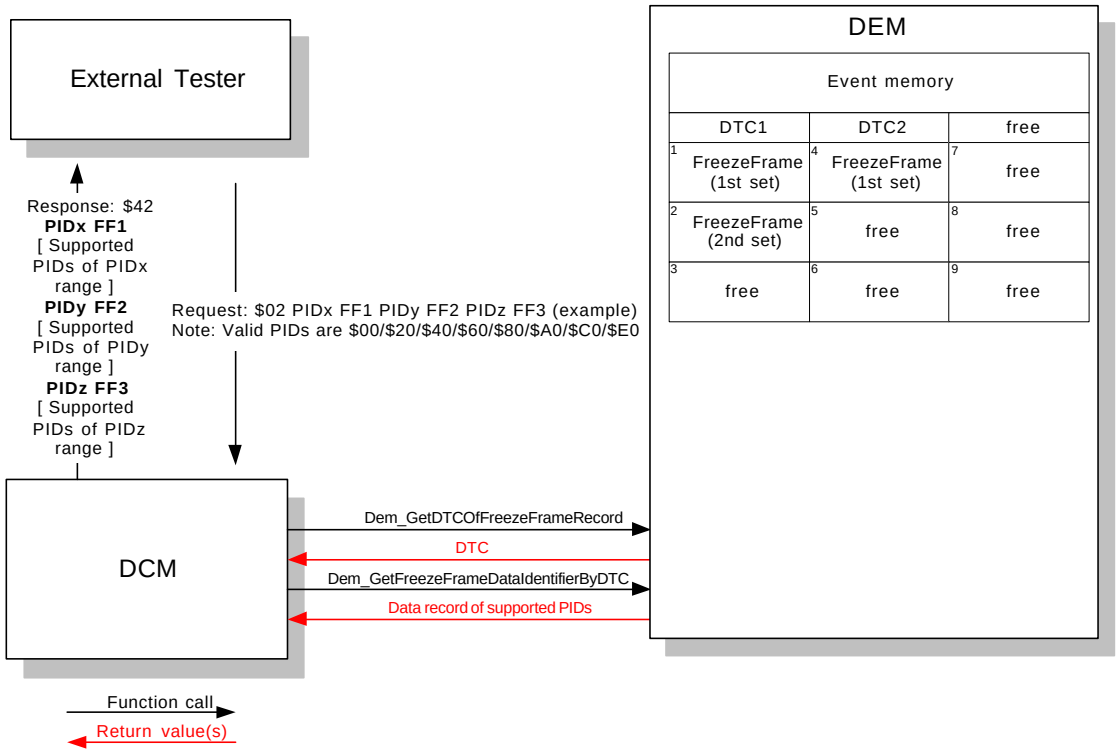


Figure 9: Request powertrain freeze frame data – Request supported PIDs of a particular Freeze Frame

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

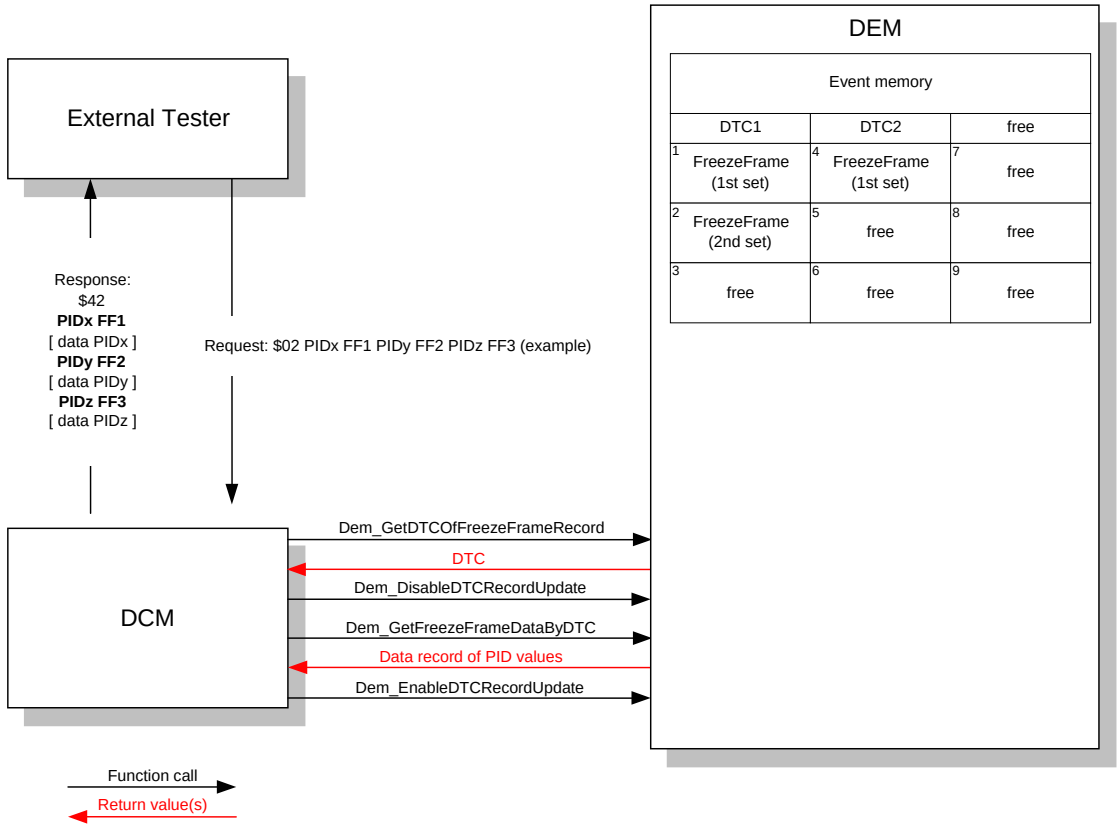


**Figure 10: Request powertrain freeze frame data –
Request data assigned to the PIDs of a particular Freeze Frame**

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_EnableDTCRecordUpdate.

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 subfuctions 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 (ReportMirrorMemoryDTCExtendedData RecordByDTCNumber)	*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, 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. After that the DSP calls Dem_SetDTCFilter to filter on the DTCs matching the requested status, memory location and DTC type.
C.3. Then the DSP calls Dem_GetNumberOfFilteredDTC to retrieve the count of filtered DTCs.
C.4. The DSP sequentially calls the Dem_GetNextFilteredDTC to retrieve the first / next DTC and the associated DTC status
C.5. 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 supporte 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

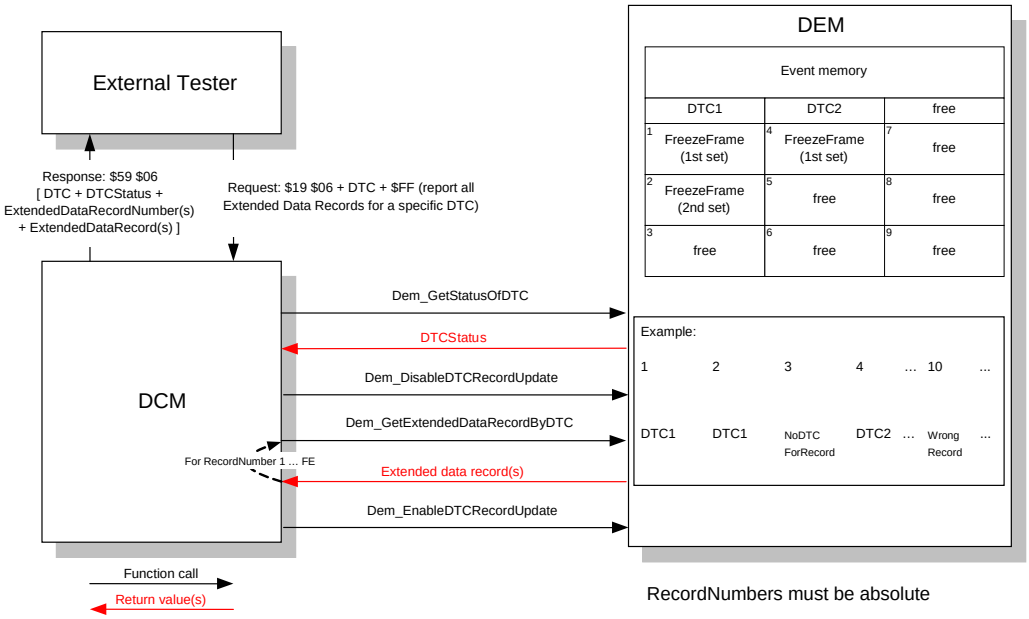


Figure 11: Request all Extended Data Records for a particular DTC

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_EnableDTCRecordUpdate.

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

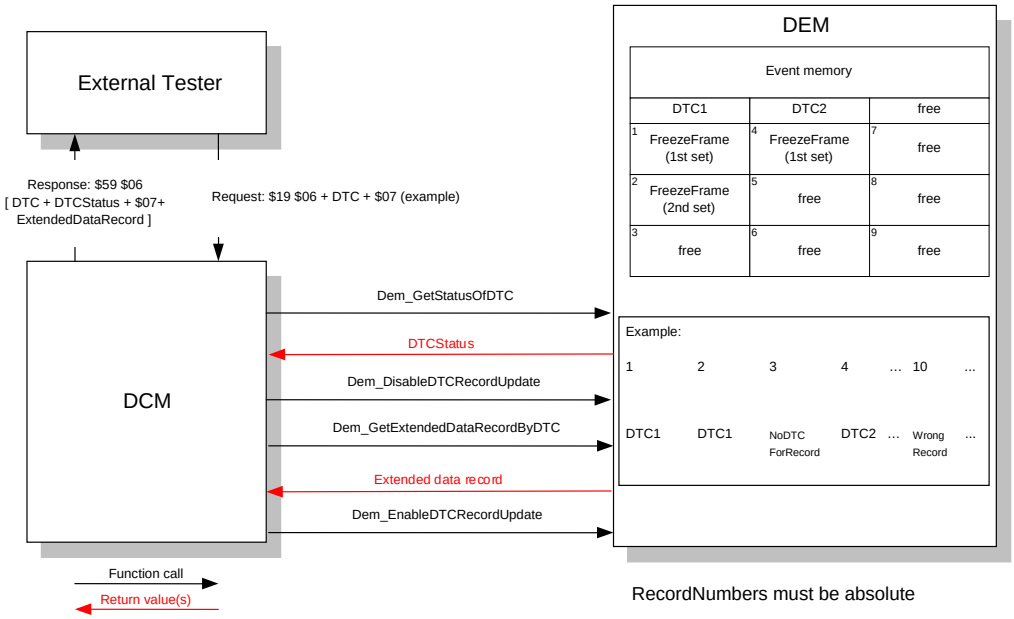


Figure 12: Request one specific Extended Data Records for a particular DTC

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

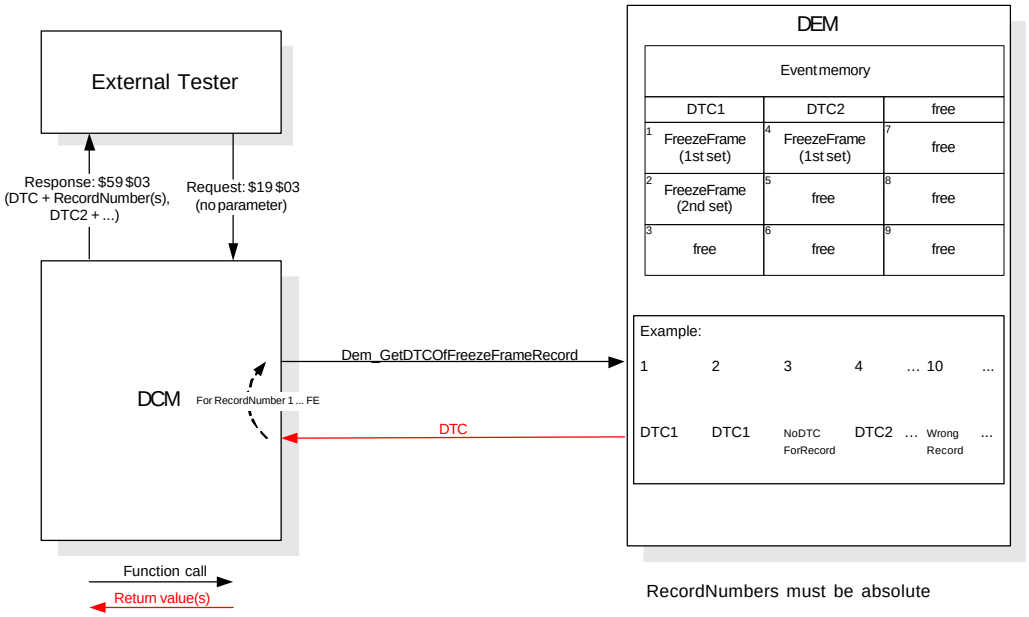


Figure 13: Request DTC Snapshot Record Identification

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

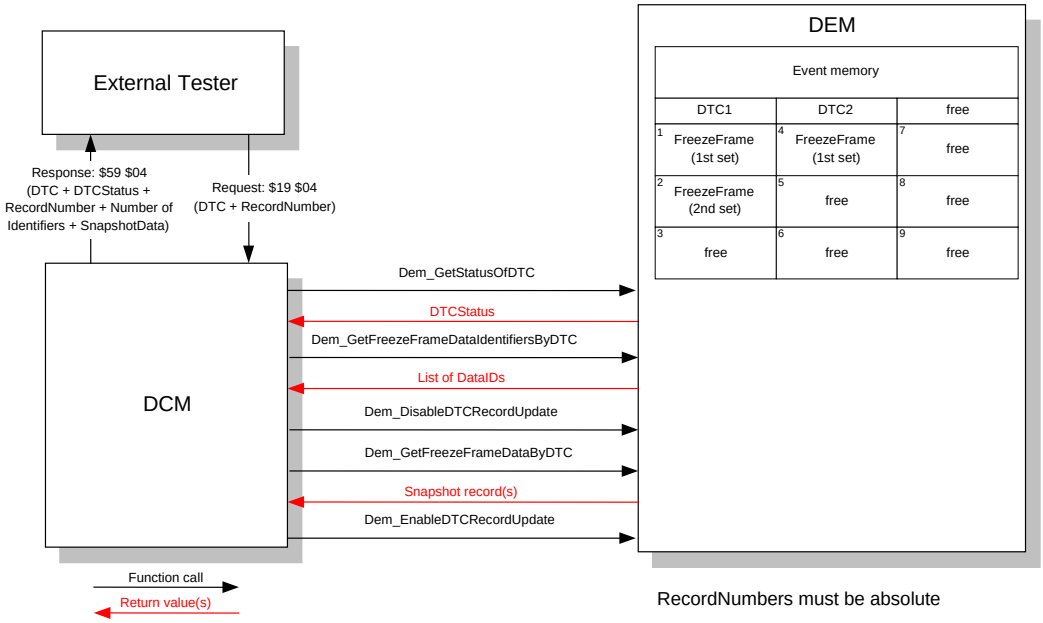


Figure 14: Request DTC Snapshot Record by DTC Number

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 calls Dem_GetFreezeFrameDataIdentifierByDTC.
H.5. The DEM returns the corresponding list of DataIDs included in the Freeze Frame the list was requested for.
H.6. 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.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

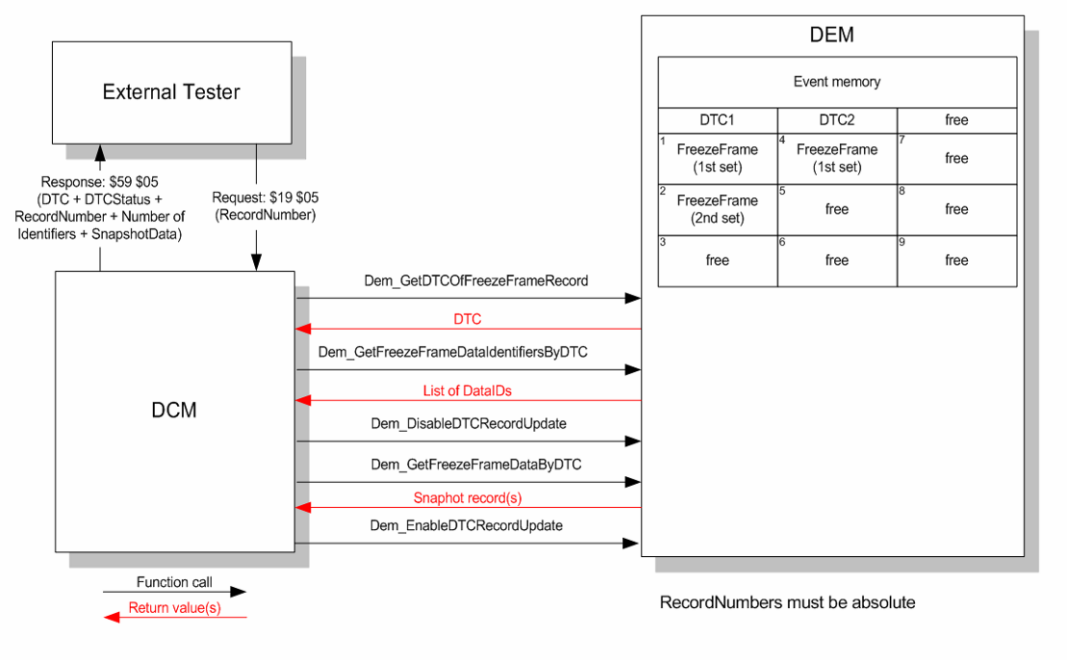


Figure 15: Request DTC Snapshot Record by Snapshot Record Number

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 calls Dem_GetFreezeFrameDataIdentifierByDTC which returns a list of DataIDs associated to the requested DTC.
I.5. 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.
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_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.

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_EnableDTCStorage is called.

7.2.3.6 Special Services

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 ***DCM_SESSION_LEVEL***).

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 ([6] chapter 8.2).
Dcm045:
Call the application callback function (see **RTE_DcmGetSesChgPermission()**) 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 **(**$P2_{ServerMax}$, $P2_{ServerMax*}$) (see configuration parameters ***DCM_SESSION_P2SERVER_MAX and DCM_SESSION_P2STRSERVER_MAX***) **(Dcm_SetSesTimingValues())**

Referenced configuration parameters can be found in chapter 10.5 DSP (Diagnostic Service Processing). 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 ***DCM_SEC_LEVEL***.

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:

- **RTE_DcmGetSeed()** (in case “request seed” is received)
- **RTE_DcmCompareKey()** (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 ***DCM_SEC_DELAY_INV_KEY***)
- ExeedNumberOfAttempts (NRC 0x36), when number of attempts to get security access exceeds (see configuration parameter ***DCM_SEC_NUM_MAX_ATT_LOCK***)
- 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 Processing).
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.

Type or error	*Relevance*	*Related error code*	*Value*
Application-Interface: Timeout	Development	DCM_E_INTERFACE_TIMEOUT	0x01
Application-Interface: Return-value out of range	Development	DCM_E_INTERFACE_VALUE_OUT_OF_RANGE	0x02
Application-Interface: Buffer Overflow	Development	DCM_E_INTERFACE_BUFFER_OVERFLOW	0x03
Application-Interface: Protocol mismatch	Development	DCM_E_INTERFACE_PROTOCOL_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 *DCM_DEV_ERROR_DETECT* (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 *DCM_DEV_ERROR_DETECT* 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 *DCM_DEV_ERROR_DETECT* 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(ModuleId, ApiId, ErrorId)
```

Where:

- `ModuleId` is the basic software module identifier. It is a uint16 constant formatted as following:
  - High byte: Module identifier of the calling module which is defined in the Specification of Module Development Error Tracer(DCM module ID = 0x35)
  - Low byte: instance identifier of the calling module, because DCM has a unique instance this byte is set to 0
- `ApiId` is a uint8 constant specified within this specification document (This is the service Id of the calling primitive).
- `ErrorId` is a uint8 constant that represents the error identifier. 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

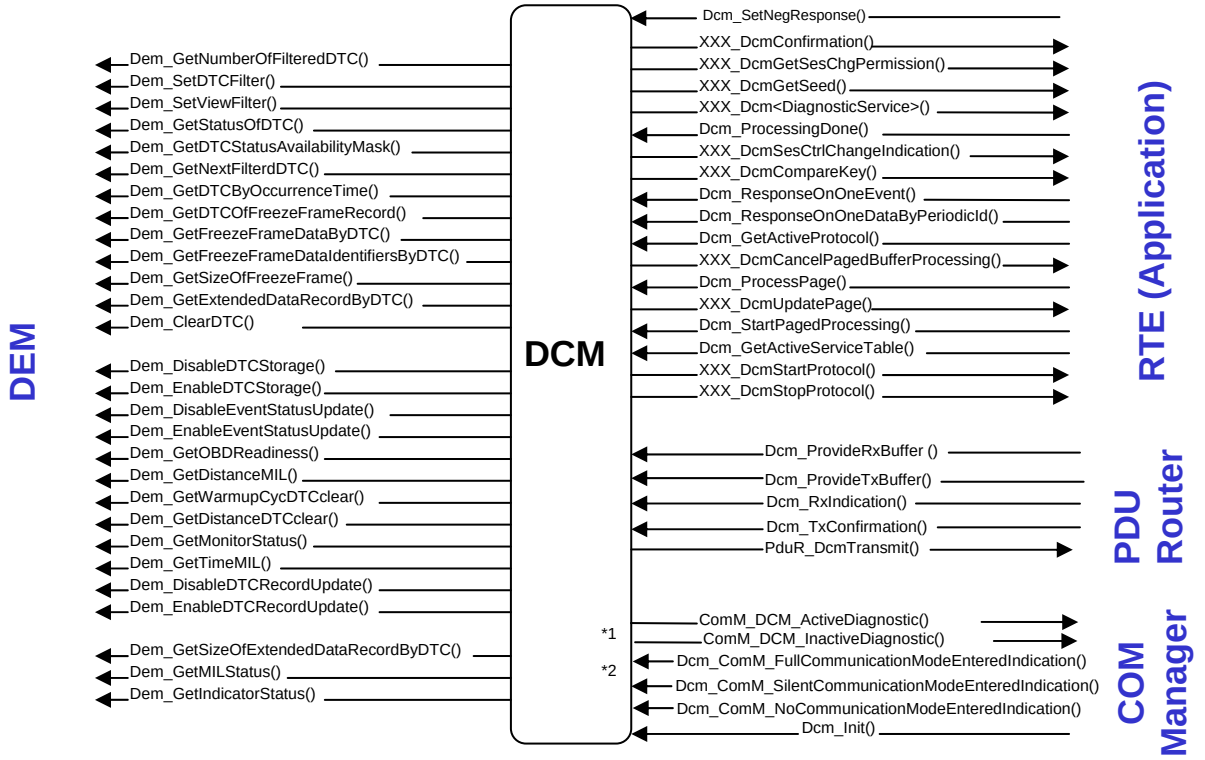
Dcm059: The same intention, logical contents or semantic shall be placed in one parameter only. (There must not be several parameters with the same intention, logical contents or semantic).

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.

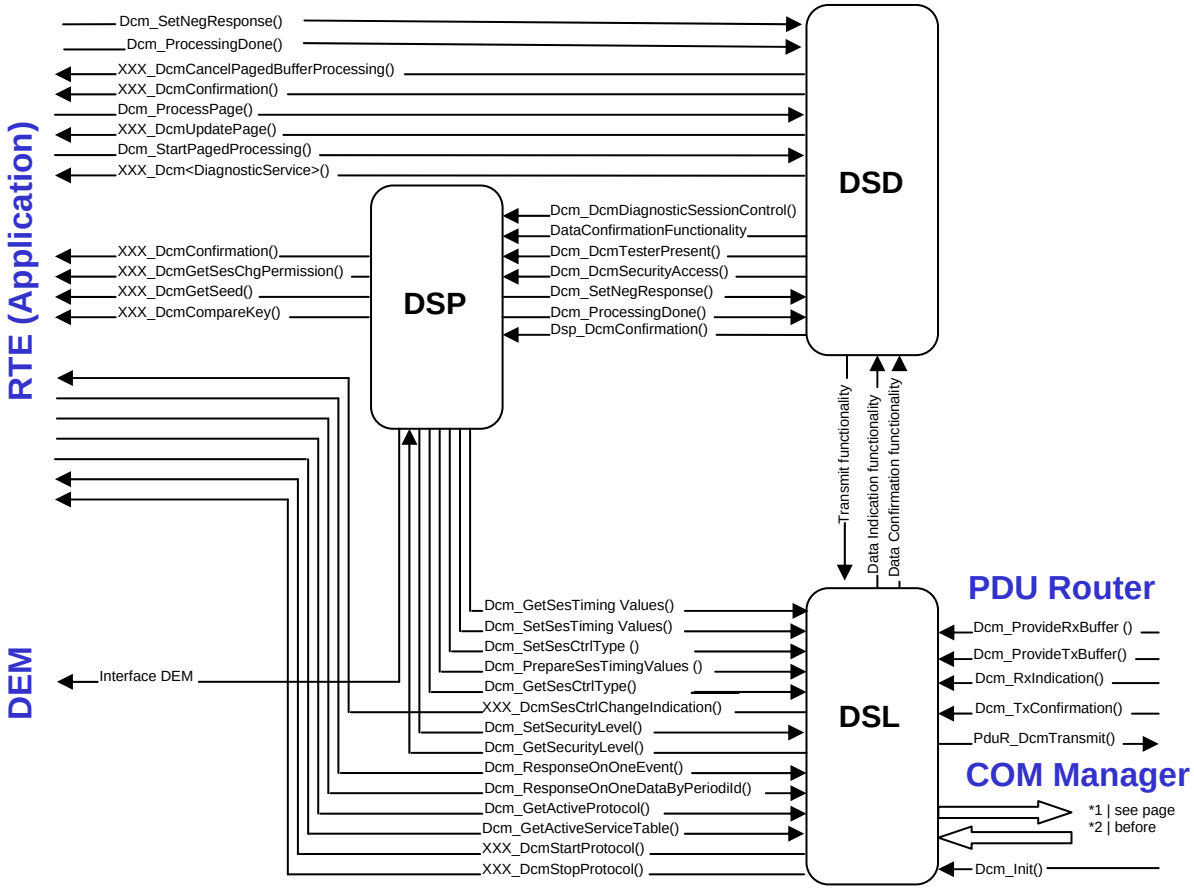
Dcm062: Each parameter shall have a list of valid values or the minimum as well as maximum values shall be specified.

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:	`uint8`	`DCM_E_OK`	0 This value is representing a successful operation.
		`DCM_E_COMPARE_KEY_FAILED`	1 ECU compares by tester requested key with own calculated key When comparison fails this definition is used. (used at API: RTE_DcmCompareKey() within processing of security access service)
		`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 (used at API: RTE_DcmGetSesChgPermission())
		`DCM_E_PROTOCOL_NOT_ALLOWED`	5 Application does not allow start of requested protocol (used at API: RTE_DcmStartProtocol())
		`DCM_E_ROE_NOT_ACCEPTED`	6 ResponseOnOneEvent request is not accepted by DCM (e.g. old ResponseOnOneEvent is not finished) (used at API: Dcm_ResponseOnOneEvent())
		`DCM_E_PERIODICID_NOT_ACCEPTED`	7 Periodic transmission request is not accepted by DCM (e.g. old Periodic transmission is not finished) (used at API: Dcm_ResponseOnOneDataByPeriodicId())
		`DCM_E_REQUEST_NOT_ACCEPTED`	8 Application rejects diagnostic request -> (used at API: RTE_DcmIndication())

		DCM_E_REQUEST_ENV_NOK	9 Diagnostic request is not allowed by application because of not fitting environmental conditions -> (used at API: RTE_DcmIndication())
Description:	Base item type to transport status information.		

8.3.1.2 Dcm_ProtocolType

Type:	typedef			
Range:	uint8	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..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	
Element:	uint8 : 1bit	reqType 0 = physical request 1 = functional request
Element:	uint8 : 1bit	suppressPosResponse 0 = no (do not suppress) 1 = yes (no positive response will be sent)
Description:	Additional information on message request.	

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	PduIdType	dcmRxPduId: Pdu identifier on which the request was received. The PduId of the request can have consequences for message processing. E. g. an OBD request will be received on the OBD PduId and will be processed slightly different than an enhanced diagnostic request received on the physical PduId.
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_SERVICENOTSUPPORTED	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_SUBFUNCTIONNOTSUPPORTED	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_BUSYREPEATREQUEST	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. 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. 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. 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.
		0x22	DCM_E_CONDITIONSNOTCORRECT	This return value indicates that the requested action will not be taken because the application prerequisite conditions are not met.

		0x24	DCM_E_REQUESTSEQUENCEERROR	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. 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.
		0x31	DCM_E_REQUESTOUTOFRANGE	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_routineIdentifer that is not supported or not supported in active session. 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.
		0x33	DCM_E_SECURITYACCESSDENIED	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. The application shall send this return value if one of the following cases occur: - 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. 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.
		0x35	DCM_E_INVALIDKEY	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.

		0x36	DCM_E_EXCEEDNUMBEROFATTEMPTS	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.
		0x37	DCM_E_REQUIREDTIMEDELAYNOTEXPIRED	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.
		0x70	DCM_E_UPLOADDOWNLOADNOTACCEPTED	This return value indicates that an attempt to upload_download to a application's memory cannot be accomplished due to some fault conditions.
		0x71	DCM_E_TRANSFERDATASUSPENDED	This return value indicates that a data transfer operation was halted due to some fault.
		0x72	DCM_E_GENERALPROGRAMMINGFAILURE	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).
		0x73	DCM_E_WRONGBLOCKSEQUENCECOUNTER	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.

		0x78	DCM_E_REQUESTCORRECTLYRECEIVED-RESPONSEPENDING	This return value is usually sent automatically by the DCM, however for special use case (e.g Flash programming) this negative response code can be used by the application. This return value indicates that the request message was received correctly, and that all parameters in the request message were valid, but the action to be performed is not yet completed and the application is not yet ready to receive another request. As soon as the requested service has been completed, the application shall send a positive response message or negative response message with a return value different from this. The negative response message with this return value may be repeated by the application until the requested service is completed and the final response message is sent. This return value might impact the application layer timing parameter values. The detailed specification shall be included in the data link specific implementation document. This return value shall only be used in a negative response message if the application will not be able to receive further request messages from the client while completing the requested diagnostic service. When this return value is used, the application shall always send a final response (positive or negative) independent of the suppressPosRspMsgIndicationBit value. A typical example where this return value may be used is when the client has sent a request message, which includes data to be programmed or erased in flash memory of the application. If the programming_erasing routine (usually executed out of RAM) is not able to support serial communication while writing to the flash memory the application shall send a negative response message with this return value. 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

		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_RPMTOOHIGH	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_RPMTOOLOW	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_ENGINERUNTIMETOOLOW	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_VEHICLESPEEDTOOHIGH	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_VEHICLESPEEDTOOLOW	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_TRANSMISSIONRANGENOTINNEUTRAL	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_TRANSMISSIONRANGENOTINGEAR	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_SHIFTERLEVERNOTINPARK	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_VOLTAGETOOHIGH	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_VOLTAGETOOLOW	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 of DiagnosticSessionControl request (see [6]))>
		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
		0x03...0x7F	<configuration dependent – only odd values are allowed (according to "securityAccessType" parameter of SecurityAccess request (see [6]))>
		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:	This service returns the version information of this module. The version information includes:	

	- Module Id - Vendor Id - Vendor specific version numbers (BSW00407). This function shall be pre compile time configurable `On/Off` by the configuration parameter: DCM_VERSION_INFO_API 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.
Caveats:	None
Configuration:	DCM_VERSION_INFO_API and configuration parameters in chapter 10.6

8.4.2 DSL (Diagnostic Session Layer) Functions

Session State Access

8.4.2.1 Dcm_GetSesCtrlType

Service name:	Dcm_GetSesCtrlType
Syntax:	`Dcm_SesCtrlType Dcm_GetSesCtrlType (void)`
Service ID:	0x06
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	Active Session Control Type value Content is according to “diagnosticSessionType” parameter of DiagnosticSessionControl request (see [6])
Description:	DCM provides with this the active session control type value
Caveats:	None
Configuration:	None

8.4.2.2 Dcm_SetSesCtrlType

Service name:	Dcm_SetSesCtrlType	
Syntax:	`void Dcm_SetSesCtrlType ( Dcm_SesCtrlType SesCtrlType )`	
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 [6])
Parameters (out):	None	
Return value:	None	
Description:	to set the new session control type value in DCM	
Caveats:	None	
Configuration:	None	

Session Timing Access

8.4.2.3 Dcm_GetSesTimingValues

Service name:	Dcm_GetSesTimingValues	
Syntax:	`void Dcm_GetSesTimingValues ( Dcm_TimerModeType TimerMode, Dcm_TimerServerType* TimerServerCurrent )`	
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	

8.4.2.4 Dcm_PrepareSesTimingValues

Service name:	Dcm_PrepareSesTimingValues
Syntax:	`Dcm_StatusType Dcm_PrepareSesTimingValues`

	(const Dcm_TimerServerType* TimerServerNew)
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

8.4.2.5 Dcm_SetSesTimingValues

Service name:	Dcm_SetSesTimingValues
Syntax:	void Dcm_SetSesTimingValues (const Dcm_TimerServerType* TimerServerNew)
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 (RTE_DcmConfirmation or 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

Security Level Access

8.4.2.6 Dcm_GetSecurityLevel

Service name:	Dcm_GetSecurityLevel
Syntax:	Dcm_SecLevelType Dcm_GetSecurityLevel (void)
Service ID:	0x0D
Sync/Async:	Synchronous

Reentrancy:	Reentrant
Parameters (in):	None
Parameters (out):	None
Return value:	Active 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 [6])
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

8.4.2.7 Dcm_SetSecurityLevel

Service name:	Dcm_SetSecurityLevel	
Syntax:	`void Dcm_SetSecurityLevel ( Dcm_SecLevelType SecurityLevel )`	
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 [6])
Parameters (out):	None	
Return value:	None	
Description:	to set the new security level value in DCM	
Caveats:	None	
Configuration:	None	

Protocol information

8.4.2.8 Dcm_GetActiveProtocol

Service name:	Dcm_GetActiveProtocol	
Syntax:	`void 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:	None	

Description:	DCM provides with this the active Protocol name.
Caveats:	None
Configuration:	DCM_PROTOCOL_ID

ROE handling

8.4.2.9 Dcm_ResponseOnOneEvent

Service name:	Dcm_ResponseOnOneEvent	
Syntax:	`Dcm_StatusType Dcm_ResponseOnOneEvent ( Dcm_MsgType MsgPtr, Dcm_MsgLenType MsgLen, PduIdType DcmRxPduId )`	
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. Application or ROE manager 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 by the RTE. A sequence diagram showing more details can be found in chapter 9.2.5 Process single event-triggered response of ResponseOnEvent	
Caveats:	None	
Configuration:	DCM_ROE_ENABLED	

Periodic transmission handling

8.4.2.10 Dcm_ResponseOnOneDataByPeriodicId

Service name:	Dcm_ResponseOnOneDataByPeriodicId
Syntax:	`Dcm_StatusType Dcm_ResponseOnOneDataByPeriodicId`

	(uint8 PeriodicId)
Service ID:	0x11
Sync/Async:	Synchronous
Reentrancy:	Not reentrant. Function may not be called reentrantly (recursively). Only one processing at one time. Application or schedule manager 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 by the RTE. 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 ReadDataByPeriodicIdentifie
Caveats:	None
Configuration:	DCM_PERIODIC_TRANS_ENABLED

8.4.3 DSD (Diagnostic Service Dispatcher)

8.4.3.1 Dcm_ProcessingDone

Service name:	Dcm_ProcessingDone
Syntax:	void Dcm_ProcessingDone (Dcm_MsgContextType* pMsgContext)
Service ID:	0x14
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	pMsgContext — Message-related information for one one diagnostic protocol identifier.
Parameters (out):	None
Return value:	None.
Description:	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 application does not have to care about any timing requirement to process a request. For diagnostic experts: Between the arrival of a request (XXX_Dcm<DiagnosticService>) and finishing the corresponding response (Dcm_Processing_Done), busy-acknowledges (negative response with response code 0x78) are sent by the DSL automatically.
Caveats:	

Configuration:	None

8.4.3.2 Dcm_StartPagedProcessing

Service name:	Dcm_StartPagedProcessing	
Syntax:	`void Dcm_StartPagedProcessing ( 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 application 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 and RTE. More information can be found in the sequence chart in chapter 9.3.6 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:	DCM_PAGEDBUFFER_ENABLED	

8.4.3.3 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:	application requests transmission of filled page More information can be found in the sequence chart in chapter 9.3.6 Process Service Request with PagedBuffer.	
Caveats:	This API is only needed at PagedBuffer handling	
Configuration:	DCM_PAGEDBUFFER_ENABLED	

8.4.3.4 Dcm_GetActiveServiceTable

Service name:	Dcm_GetActiveServiceTable
Syntax:	`void Dcm_GetActiveServiceTable (uint8* ActiveServiceTable)`
Service ID:	0x1C
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (out):	`ActiveServiceTable` Active service table
Return value:	None
Description:	Configuration allows creation of multiple service identifier tables. It is now possible, that a service interpreter (provided inside the Central Diagnostic Software component) allows also configuration of multiple service interpreter tables (e.g table for ReadDataByIdentifier processing). The selection of service interpreter table can be upon value of active service table.
Caveats:	None
Configuration:	DCM_SIDTAB_ID

8.4.4 Dcm_SetNegResponse

Service name:	Dcm_SetNegResponse	

Syntax:	`void Dcm_SetNegResponse`
	`(`
	`Dcm_MsgContextType* pMsgContext,`
	`Dcm_NegativeResponseCodeType ErrorCode`
	`)`	

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:	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. The transmission of the negative response will be performed after calling Dcm_ProcessingDone. For a list of negative response codes supported, please refer chapter 8.3.1.9 Dcm_NegativeResponseCode	
Caveats:		
Configuration:	None	

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:	`BufReq_ReturnType Dcm_ProvideRxBuffer ( PduIdType DcmRxPduId, PduLengthType TpSduLength, PduInfoType** PduInfoPtr )`	
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:	By this service the DCM is requested to provide a buffer for a transport layer on first frame or single frame reception. 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 `**PduInfoPtr.` By this service the receiver (e.g. DCM) is also informed implicitly about a first frame reception or a single frame reception. If a return value not equal to BUFREQ_E_OK is returned, the values of the out Parameters are not specified and shall not be evaluated.	
Caveats:	After returning a valid buffer, the DCM must not access this buffer unless: • it is being notified by the service Dcm_RxIndication about the successful	

	reception (indication), or • it is being notified by the service Dcm_RxIndication the reception was aborted (error indication). The transport protocol filling the provided buffer will also set the length information contained in `PduInfoPtr` to the number of bytes that are valid in this buffer.
Configuration:	None

8.5.1.2 Dcm_RxIndication

Service name:	Dcm_RxIndication	
Syntax:	`void Dcm_RxIndication ( PduIdType DcmRxPduId, NotifResultType Result )`	
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: • 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): • 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:	`BufReq_ReturnType Dcm_ProvideTxBuffer ( PduIdType DcmTxPduId, PduInfoType**PduInfoPtr, 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 `PduR_DcmTransmit()`.
	`Length`	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. The value of this parameter shall not exceed the number of Bytes still to be sent. A length of zero indicates that the length of the buffer can be of arbitrary size (larger than zero). The `Length` parameter is expected by DCM to be always 0 (zero).
Parameters (out):	`PduInfoPtr`	Pointer to pointer to PduInfoStructure containing data pointer and length of a transmit buffer. This length must not be smaller than the length given by `MinLength`.
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:	By this service the DCM is requested to provide a buffer containing data to be transmitted via a transport protocol. 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 (`Length`). If the returned buffer is smaller than the length requested to transmit within the service `PduR_DcmTransmit()` the DCM will be requested by this service to provide another buffer after the data of the current buffer have been transmitted. If the DCM can provide a buffer of the size `Length` or larger, this service returns BUFREQ_OK. If the DCM cannot provide a buffer of the size `Length`, or larger (except `Length` is equal zero) an the service returns with BUFREQ_E_NOT_OK. If the DCMtemporarily cannot provide a buffer of the requested length, the service returns with BUFREQ_E_BUSY. This service has to be called again during next processing of the MainFunction of the transport protocol	
Caveats:	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 `DcmTxPduId` or it is notified about the successful transmission (confirmation) or it is notified by	

	an error indicating that the transmission was aborted (error indication). If this service returns BUFREG_E_NOT_OK the transmit requests issued by calling the service PduR_DcmTransmit is still not finished. A final confirmation (indicating an error) is required to finish this service and to start a subsequent request by calling PduR_DcmTransmit. So it is up to the transport protocol if the transmission is aborted on BUFREQ_E_NOT_OK or not.
Configuration:	None

8.5.1.4 Dcm_TxConfirmation

Service name:	Dcm_TxConfirmation	
Syntax:	`void Dcm_TxConfirmation ( PduIdType DcmTxPduId, NotifResultType Result )`	
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: • 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.
Parameters (out):	None	
Return value:	None	
Description:	This function is called by the lower layer (in general the PDU Router): • with Result = NTFRSLT_OK after the complete DCM I-PDU has successfully been transmitted, i.e. at the very end ofthe 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. <Further detailed description>	
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:	DCM_TASK_TIME

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***
PduR_DcmTransmit	<PduR>	forwards the diagnostic message to the transmitting layers (e.g. CAN TP)
Dem_GetNumberOfFilteredDTC	<Dem>	DEM shall return the number of filtered DTCs (for the set filter)
Dem_GetNextFilteredDTCAndFDC	<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]
Dem_SetDTCFilter	<Dem>	DCM can set the DTC filter
Dem_SetViewFilter	<Dem>	DCM can set the VIEW filter
Dem_GetStatusOfDTC	<Dem>	DEM shall return the state of the

		requested DTC
Dem_GetDTCStatusAvailabilityMask	<Dem>	DEM shall return the StatusAvailabilityMask of the requested DTC
Dem_GetNextFilterdDTC	<Dem>	DEM shall return next filtered DTC (will be called in a loop)
Dem_GetDTCByOccurrenceTime	<Dem>	DEM shall return DTCs by occurrence time
Dem_GetDTCOfFreezeFrameRecord	<Dem>	DEM shall the DTC associated with the Freeze Frame selected via its absolute record number.
Dem_GetSizeOfFreezeFrame	<Dem>	DEM shall return the size of the requested FreezeFrame
Dem_GetFreezeFrameDataByDTC	<Dem>	DEM shall return FreezeFrame for requested DTCs
Dem_GetFreezeFrameDataIdentifierByDTC	<Dem>	DEM shall return FreezeFrame Data Identifiers for requested DTCs
Dem_GetExtendedDataRecordByDTC	<Dem>	DEM shall return extended data record for requested DTCs
Dem_ClearDTC	<Dem>	DEM shall erase requested DTCs
Dem_DisableDTCStorage	<Dem>	DEM shall disable the DTC storage
Dem_EnableDTCStorage	<Dem>	DEM shall enable the DTC storage
Dem_DisableEventStatusUpdate	<Dem>	DEM shall disable the Event Status Update
Dem_EnableEventStatusUpdate	<Dem>	DEM shall enable the Event Status Update
Dem_GetOBDReadiness	<Dem>	DEM shall return the PID01h of ISO15031-5 Data B/C/D information
Dem_GetDistanceMIL	<Dem>	DEM shall return PID21h of ISO15031-5 Data A/B information
Dem_GetWarmupCycleDTCclear	<Dem>	DEM shall return PID30h of ISO15031-5 Data A information
Dem_GetDistanceDTCclear	<Dem>	DEM shall return PID31h of ISO15031-5 Data A/B information
Dem_GetMonitorStatus	<Dem>	DEM shall return PID41h of ISO15031-5 Data B/C/D information
Dem_GetTimeMIL	<Dem>	DEM shall return PID4Dh of ISO15031-5 Data A/B information
Dem_DisableDTCRecordUpdate	<Dem>	DEM shall disable the DTC Record Update
Dem_EnableDTCRecordUpdate	<Dem>	DEM shall enable the DTC Record Update
Dem_GetSizeOfExtendedDataRecordByDTC	<Dem>	DEM shall return size of extended data record for requested DTCs
Dem_GetMILStatus	<Dem>	DEM shall return MIL-Status required for ISO15031-5
Dem_GetIndicatorStatus	<Dem>	
Dem_GetTimeDTCclear	<Dem>	DEM shall return PID4Eh of ISO15031-5 Data A/B information
ComM_DCM_ActiveDiagnostic	<ComM>	With this function the DCM indicates the start of an Diagnostic communication
ComM_DCM_InactiveDiagnostic	<ComM>	With this function the DCM indicates the stop of an Diagnostic communication
SchM_ActMainFunction_DCM	<SchM>	The DCM module implementation invokes the SchM_ActMainFunction API to trigger the activation of a corresponding main processing function.

SchM_CancelMainFunction_DCM	<SchM>	The DCM module invokes the SchM_CancelMainFunction API to trigger the cancelation of the requested activation of a corresponding main processing function.

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	DCM_DEV_ERROR_DETECT

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 RTE_DcmGetSesChgPermission

Service name:	RTE_DcmGetSesChgPermission	
Syntax:	`Dcm_StatusType RTE_DcmGetSesChgPermission ( Dcm_SesCtrlType SesCtrlTypeActive, Dcm_SesCtrlType SesCtrlTypeNew )`	
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	
Description:	Request to application, to check if the conditions allows to start 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 RTE_DcmSesCtrlChangeIndication

Service name:	RTE_DcmSesCtrlChangeIndication	
Syntax:	`void RTE_DcmSesCtrlChangeIndication ( 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:	None	
Description:	Indication to application, that value of active session control is changed. This change was either initiated by “DiagnosticSessionControl” command or $S3_{Server}$ 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 RTE_DcmStartProtocol

Service name:	RTE_DcmStartProtocol	
Syntax:	`Dcm_StatusType RTE_DcmStartProtocol ( 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_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:	DCM_PROTOCOL_ID	

8.7.3.1.4 RTE_DcmStopProtocol

Service name:	RTE_DcmStopProtocol
Syntax:	`void RTE_DcmStopProtocol ( Dcm_MsgContextType* pMsgContext`

	)	
Service ID:	0x20	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant	
Parameters (in):	`pMsgContext`	Message-related information for one diagnostic protocol identifier. Message context of preempted request
Parameters (out):	None	
Return value:	None	
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	

8.7.3.1.5 RTE_DcmConfirmation

Service name:	RTE_DcmConfirmation	

Syntax:	`void RTE_DcmConfirmation`
	`(`
	`  Dcm_IdContextType idContext,`
	`  PduIdType dcmRxPduId,`
	`  Dcm_ConfirmationStatusType status`
	`)`	

Service ID:	0x16	
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 responses was sent before.
Parameters (out):	None	
Return value:	None	
Description:	(see: Dsp_DcmConfirmation) This function confirms the successful transmission or a transmission error of a diagnostic service. This is the right time to perform any application state transitions. The idContext and the dcmRxPduId are 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. This function is only called if processing of the service itself was done within the	

	application.
Caveats:	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.
Configuration:	None

8.7.3.1.6 RTE_DcmIndication

Service name:	RTE_DcmIndication	
Syntax:	`Dcm_StatusType RTE_DcmIndication ( Dcm_MsgContextType* pMsgContext, uint8 SID )`	
Service ID:	0x17	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	`pMsgContext`	Message-related information for one diagnostic protocol identifier.
	`SID`	Value of service identifier
Parameters (out):	None	
Return value:	DCM_E_OK: diagnostic request accepted DCM_E_REQUEST_NOT_ACCEPTED: application rejects diagnostic request -> DSD will ignore diagnostic request, no further processing of request, no response will be send DCM_E_REQUEST_ENV_NOK: Environmental conditions are not ok -> DSD will generate negative response with NRC 0x22 (ConditionsNotCorrect)	
Description:	This function indicates the successful reception of a new request to application and it is called right before processing the diagnostic service XXX_Dcm<DiagnosticService>. 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:	DCM_REQUEST_INDICATION_ENABLED	

8.7.3.1.7 RTE_DcmUpdatePage

Service name:	RTE_DcmUpdatePage
Syntax:	`void RTE_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, application is requested to fill next page. Page buffersize and Page buffer is given to application More information can be found in the sequence chart in chapter 9.3.6 Process Service Request with PagedBuffer.	
Caveats:	This API is only needed at PagedBuffer handling	
Configuration:	DCM_PAGEDBUFFER_ENABLED	

8.7.3.1.8 RTE_DcmCancelPagedBufferProcessing

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

8.7.3.2 Interface for special services

8.7.3.2.1 RTE_DcmGetSeed

Service name:	RTE_DcmGetSeed	

Syntax:	`void RTE_DcmGetSeed`
	`(`
	`Dcm_SecLevelType SecurityLevel,`
	`uint8 SeedLen,`
	`uint8* const Seed`
	`)`	

Service ID:	0x1D	
Sync/Async:	Asynchronous	
Reentrancy:	Non-Reentrant	
Parameters (in):	`SecurityLevel`	Requested security level Conversion formula to calculate SecurityLevel out of tester requested SecurityAccessType parameter: SecurityLevel = (SecurityAccessType + 1) / 2
	`SeedLen`	Seed length as configured for access type (DCM_SEC_NUM_SEED)

Parameters (out):	`Seed`	Pointer for provided seed
Return value:	None	
Description:	Request to application for provision of seed value	
Caveats:	None	
Configuration:	DCM_SEC_NUM_SEED	

8.7.3.2.2 RTE_DcmCompareKey

Service name:	RTE_DcmCompareKey	
Syntax:	`Dcm_StatusType RTE_DcmCompareKey ( Dcm_SecLevelType SecurityLevel, uint8 KeyLen, const uint8* const Key )`	
Service ID:	0x1E	
Sync/Async:	Asynchronous	
Reentrancy:	Non-Reentrant	
Parameters (in):	`SecurityLevel`	Requested security level Conversion formula to calculate SecurityLevel out of tester requested SecurityAccessType parameter: SecurityLevel = SecurityAccessType / 2
	`KeyLen`	Key length as configured for access type (DCM_SEC_NUM_KEY)
	`Key`	Pointer to Key, which needs to be compared
Parameters (out):	None	
Return value:	DCM_E_OK: compare was successful DCM_E_COMPARE_KEY_FAILED: compare failed	
Description:	Request to application for comparing key	
Caveats:	None	
Configuration:	DCM_SEC_NUM_KEY	

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 names of these kind of interfaces is not fixed because they are configurable.

8.7.4.1 DSD (Diagnostic Service Dispatcher)

8.7.4.1.1 XXX_Dcm<DiagnosticService>

Service name:	XXX_Dcm<DiagnosticService>
Syntax:	`void XXX_Dcm<DiagnosticService> ( Dcm_MsgContextType* pMsgContext )`
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:	Callback function. DCM shall call this callback function as soon as valid message is received on relevant DcmRxPduId. The usecase of multiple diagnostic protocols will be possible by using different arguments and the function shall be programmed in a way at it is reentrant. This feature is not supported for Version2. Caller (lower layer) is responsible for the lifetime of the argument pMsgContext. The application does not have to care about any timing requirement to process a request
Caveats:	None
Configuration:	Only available if diagnostic corresponding protocol service is supported. DCM_SIDTAB_FUNCTIONPOINTER

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 XXX_Dcm*Servicename* for each supported diagnostic service identifier.

XXX can have the following content:

- XXX ≠ Dcm=> the function is realized in the “Central Diagnostic SW Component”
- XXX = Dcm=> the function is realized in the DCM Part DSP

Service Identifier	***Function name examples for <DiagnosticService>***
0x10	XXX_DcmDiagnosticSessionControl
0x11	XXX_DcmECUReset
0x27	XXX_DcmSecurityAccess
0x28	XXX_DcmCommunicationControl
0x3E	XXX_DcmTesterPresent
0x83	XXX_DcmAccessTimingParameter
0x84	XXX_DcmSecuredDataTransmission
0x85	XXX_DcmControlDTCSetting
0x86	XXX_DcmResponseOnEvent
0x87	XXX_DcmLinkControl
0x22	XXX_DcmReadDataByIdentifier
0x23	XXX_DcmReadMemoryByAddress
0x24	XXX_DcmReadScalingDataByIdentifier
0x2A	XXX_DcmReadDataByPeriodicIdentifier
0x2C	XXX_DcmDynamicallyDefineDataIdentifier
0x2E	XXX_DcmWriteDataByIdentifier
0x3D	XXX_DcmWriteMemoryByAddress
0x14	XXX_DcmClearDiagnosticInformation
0x19	XXX_DcmReadDTCInformation
0x2F	XXX_DcmInputOutputControlByIdentifier

0x31	XXX_DcmRoutineControl
0x34	XXX_DcmRequestDownload
0x35	XXX_DcmRequestUpload
0x36	XXX_DcmTransferData
0x37	XXX_DcmRequestTransferExit

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

ISO15031-5

Service Identifier	*Function name examples for <DiagnosticService>*
0x01	XXX_DcmRequestCurrentDiagnosticData
0x02	XXX_DcmRequestFreezeFrameData
0x03	XXX_DcmRequestEmissionRelatedDTC
0x04	XXX_DcmClearEmissionRelatedInformation
0x05	XXX_DcmRequestOxySenMoniTestResults
0x06	XXX_DcmRequestOnBoardMoniTestResults
0x07	XXX_DcmReqEmissionRelDTCDrivCyc
0x08	XXX_DcmRequestControl
0x09	XXX_DcmRequestVehicleInformation

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

8.8 Proposed internal interfaces

8.8.1 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:	This function confirms the successful transmission or a transmission error of a diagnostic service. This is the right time to perform any application state transitions. The idContext and the dcmRxPduIdare 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. The DSP receives this confirmation. If this service was not processed inside DSP, it shall defer this confirmation to the application.
Caveats:	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. 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).
Configuration:	None

8.8.2 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.

8.8.3 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.

8.8.4 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 Diagnostic Communication Manager (DCM) scheduler

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 connected over the communication network (e.g. in the development, manufacturing or service).

8.10.1 Specification of the Ports and Port Interfaces for Diagnostic Services

Parts of DCM activities are currently deported in an application software component above the RTE. This means, in term of implementation, that such software component, named “Central Diagnostic SW-C”, must be unique on a local ECU and not accessible beyond the internal ECU communication. This prerequisite only impacts service port interfaces of the Central Diagnostic.
This subchapter specifies the ports and port interfaces which are needed in order to operate the DCM functionality over the VFB.

RTE has the double job to map appropriate information from Central Diagnostic interfaces, typed as services (client) to the ports of the DCM and map the communication from the DCM, seen as client, by generating the correct port interface of the Central Diagnostic.

8.10.1.1 General Approach

As previously mentioned, the Central Diagnostic assumes both service mechanisms, client and server, towards the BSW modules DCM and both communication paradigms.
The Central Diagnostic must support several activities according to requests from all DCM subcomponents and communication with them:

- Diagnostic Session Layer which provides the name of the active diagnostic protocol such OBD or any classic diagnostic protocol, and further it has to support the handling information of one diagnostic communication over the bus.
- Diagnostic Service Dispatcher which leads the dispatching of any diagnostic service request. This service is numbered and configured to address any kind of diagnostic proprietary (i.e.: OBD). Part of services request treatment are hosted by the Central Diagnostic, this means that the DCM must be configured in a later step (i.e.: integration, after RTE generation) to make the connection between the DCM table configuration SERVICE_IDENTIFIER_TABLE.
  Further, the Central Diagnostic, in order to provide the response to a request, has to fulfill the buffer addressed and handled by DCM BSW Module. A page is processed only by a CANTP acknowledgement or in specific case by the PDU ROUTER.
- Diagnostic Service Processor which leads, for instance, special services such the requiring of a seed value mandatory to handle a security level during

diagnostic communication. Further, the Central Diagnostic also support events related to the diagnostic communication session as the consistency between the request and its status.

8.10.1.2 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 `PduIdType` used in the interfaces refers to the Specification of Communication Stack Types [SCOMTypes].

The data type `MessageAdditionalInformation`, which adds information on a message request, which will be defined by the SW-C description in the following way:

```
RecordType MessageAdditionalInformation {
IntegerType Type;
IntegerType SuppressPositiveResponse;
};
```

The data type `StatusType`, which indicates the transport status information, which will be defined by the SW-C description in the following way:

```
PrimitiveTypeWithSemantics ReturnStatusType {
     IntegerType {
          LOWER-LIMIT = 0
          UPPER-LIMIT = 9
     };
     0 -> E_OK
     1 -> DCM_E_COMPARE_KEY_FAILED
     2 -> DCM_E_TI_PREPARE_LIMITS
     3 -> DCM_E_TI_PREPARE_INCONSTENT
     4 -> DCM_E_SESSION_NOT_ALLOWED
     5 -> DCM_E_PROTOCOL_NOT_ALLOWED
     6 -> DCM_E_ROE_NOT_ACCEPTED
     7 -> DCM_E_PERIODICID_NOT_ACCEPTED
     8 -> DCM_E_REQUEST_NOT_ACCEPTED
     9 -> DCM_E_REQUEST_ENV_NOK
};
```

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:

```
PrimitiveTypeWithSemantics ProtocolType {
     IntegerType {
          LOWER-LIMIT = 0
          UPPER-LIMIT = 255
     };
     0 -> DCM_OBD_ON_CAN
     1 -> DCM_UDS_ON_CAN
     2 -> DCM_UDS_ON_FLEXRAY
     3 -> Reserved for further AUTOSAR implementation (up to
239)
     240 -> Reserved for SW supplier specific (up to 255)
};
```

The data type MessagePtrType, which indicates the base type for diagnostic message like request positive or negative response, this pointer contains the address to the data transmitted to the DCM to fulfill the buffer, which will be defined by the SW-C description in the following way:

```
ArrayType MessagePtrType
{
     elementType = Uint8;
     maxNumberOfElements > 0;
};
```

The data type MessageLengthType, which indicates the length of diagnostic message. The maximum length is dependent of the underlying transport protocol/media (ie: the maximum message length for CAN Transport Layer is 4095 bytes), which will be defined by the SW-C description in the following way:

```
IntegerType Uint16 MessageLengthType;
```

The data type IDContextType, which indicates the context identifier which can be used to determine the relation between request and response confirmation, will be defined by the SW-C description in the following way:

```
IntergerType Uint8 IDContextType;
```

The data type MessageContext, which indicates all informations which are necessary to process a diagnostic message from request to response and response confirmation will be defined by the SW-C description in the following way:

```
RecordType MessageContext {
       MessageType           RequestData;
       MessageLengthType     RequestDataLength;
       MessagePtrType        ResponseData;
       MessageLengthType     ResponseDataLength;
       MessageAdditionalInformationType MessageAddInformation;
       MessageLengthType     ResponseMaxDataLentgh;
       IDContextType         IdContext;
       ReceivePDUId          DcmRxPDUId;
       };
```

The data type NegativeResponseCodeType, which represents all Negative Response Codes allowed that a AUTOSAR SW-C shall used in case of negative response. will be defined by the SW-C description in the following way:

```
PrimitiveTypeWithSemantics NegativeResponseCodeType {
     IntegerType {
          LOWER-LIMIT = 16
          UPPER-LIMIT = 137
     };
     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_SUBFUNCTIONNOTSUPPORTEDINACTIVESESSION
     0x7F -> DCM_E_SERVICENOTSUPPORTEDINACTIVESESSION
     0x81 -> DCM_E_RPMTOOHIGH
     0x82 -> DCM_E_RPMTOOLOW
     0x83 -> DCM_E_ENGINEISRUNNING
     0x84 -> DCM_E_ENGINEISNOTRUNNING
     0x85 -> DCM_E_ENGINERUNTIMETOOLOW
     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_VOLTAGETOOLOW
};
```

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

```
PrimitiveTypeWithSemantics SessionControlType {
     IntegerType {
          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 SecurityLevelType, which defines the security level type, which will be defined by the SW-C description in the following way:

```
PrimitiveTypeWithSemantics SecurityLevelType {
     IntegerType {
          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
};
```

The data type ConfirmationStatusType, which defines additional information on message request, which will be defined by the SW-C description in the following way:

```
PrimitiveTypeWithSemantics ConfirmationStatusType {
     IntegerType {
          LOWER-LIMIT = 0
          UPPER-LIMIT = 3
     };
     0 -> DCM_RES_POS_OK
     1 -> DCM_RES_POS_NOT_OK
     2 -> DCM_RES_NEG_OK
     3 -> DCM_RES_NEG_NOT_OK
};
```

For the use of Central Diagnostic callback we need a type SeedPtrType, which points to the SEED address, which will be defined by the SW-C description in the following way:

```
ArrayType SeedPtrType
{
     elementType = Uint8;
     maxNumberOfElements > 0;
};
```

For the use of Central Diagnostic callback we need a type KeyPtrType, which points to the KEY address, the SEED is necessary to compute the KEY, which will be defined by the SW-C description in the following way:

```
ArrayType KeyPtrType
{
     elementType = Uint8;
     maxNumberOfElements > 0;
};
```

8.10.1.3 Port Interface of Diagnostic Services

Diagnostic Communication Manager services operations are put all in together in one single port interface, for XML optimization.
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.

```
ClientServerInterface DCMService {

     PossibleErrors {
               E_OK = 0,
               E_NOT_OK = 1,
          DCM_E_COMPARE_KEY_FAILED = 2,
          DCM_E_TI_PREPARE_LIMITS = 3,
          DCM_E_TI_PREPARE_INCONSTENT = 4,
          DCM_E_SESSION_NOT_ALLOWED = 5,
          DCM_E_PROTOCOL_NOT_ALLOWED = 6,
          DCM_E_ROE_NOT_ACCEPTED = 7,
          DCM_E_PERIODICID_NOT_ACCEPTED = 8,
          DCM_E_REQUEST_NOT_ACCEPTED = 9,
          DCM_E_REQUEST_ENV_NOK = 10,
          };


     SetNegResponse (IN MessageContextType pMsgContext, IN
     uint8 ErrorCode);

     ProcessingDone (IN MessageContextType pMsgContext);

     GetActiveProtocol(OUT ProtocolType ActiveProtocol);
```

```
    //Some DCM APIs should be used in case of specific uses
    //of UDS services. It is not clear and decided neither to
    //derive this in separate Client Server port interface nor
    //to include in the client server DCMService port
    //interface.
    ResponseOnOneEvent(IN MessageType MsgPtr, IN
    MessageLenthType MsgLen, IN PduIdType DcmRxPduId, ERR{
    DCM_E_ROE_NOT_ACCEPTED});

    ResponseOnOneDataByPeriodicId (IN uint8 PeriodicId, ERR{
    DCM_E_PERIODICID_NOT_ACCEPTED});

    //The Central Diagnostic has to fulfill all request
    //information within a buffer normally handled by the SW-C
    //itself (RTE).

    ProcessPage (IN MessageLengthType FilledPageLen);

    StartPagedProcessing (IN MessageContextType pMsgContext);
};
```

8.10.1.4 Port Interface for Central Diagnostic CallBack treatment

The port interface DiagnosticService is specific per Service ID (SID) (see [7]) it means that one port interface must be generated per service request. But from a Sender Receiver interface description point of view it could be summarized with only one operation related to the service ID. Thus the generated RTE will provide one API per DCM services. It implies a new configuration step of the DCM.

Such approach invites to configure the DCM subcomponent DSD to address the appropriate API which provides the Service ID request even if this services is implemented in the DCM BSW Module or the Central Diagnostic.

Service Identifier	***Function name examples for <DiagnosticService>***
0x10	XXX_DcmDiagnosticSessionControl
0x11	XXX_DcmECUReset
0x27	XXX_DcmSecurityAccess
0x28	XXX_DcmCommunicationControl
0x3E	XXX_DcmTesterPresent
0x83	XXX_DcmAccessTimingParameter
0x84	XXX_DcmSecuredDataTransmission
0x85	XXX_DcmControlDTCSetting
0x86	XXX_DcmResponseOnEvent
0x87	XXX_DcmLinkControl

Extract list of function naming examples for request of ISO14229-1

```
SenderReceiverInterface CallBackDCMRequestServices {
     DiagnosticService (IN RequestID, IN MessageContextType
pMsgContext);

     Confirmation (IN IDContextType idContext, IN PduIdType
dcmRxPduId, IN ConfirmationStatusType status);

     CancelPagedBufferProcessing ();

     StopProtocol (IN MessageContextType pMsgContext);
};



ClientServerInterface CallBackReqTreatment {
     GetSeed (IN SecLevelType SecurityLevel,IN Uint8 SeedLen,
IN SeedPtrType);

GetSesChgPermission (IN SesCtrlType SesCtrlTypeActive, IN
SesCtrlType SesCtrlTypeNew, ERR{DCM_E_SESSION_NOT_ALLOWED});

SesCtrlChangeIndication (IN SesCtrlType SesCtrlTypeOld, IN
SesCtrlType SesCtrlTypeNew);

UpdatePage (IN MessageType PageBufPtr, IN MessageType
PageLen);

     Indication (IN MessageContextType pMsgContext, IN Uint8
SID, ERR{DCM_E_REQUEST_NOT_ACCEPTED, DCM_E_REQUEST_ENV_NOK});

CompareKey (IN SecurityLevelType SecurityLevel, IN Uint8
KeyLen, IN KeyPtrType Key, ERR{DCM_E_COMPARE_KEY_FAILED});

     StartProtocol (IN ProtocolType ProtocolID,
ERR{DCM_E_PROTOCOL_NOT_ALLOWED}})
};
```

9 Sequence diagrams

9.1 Overview

For a overview please refer to chapter7.1.1 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 show on the next sequence diagrams.

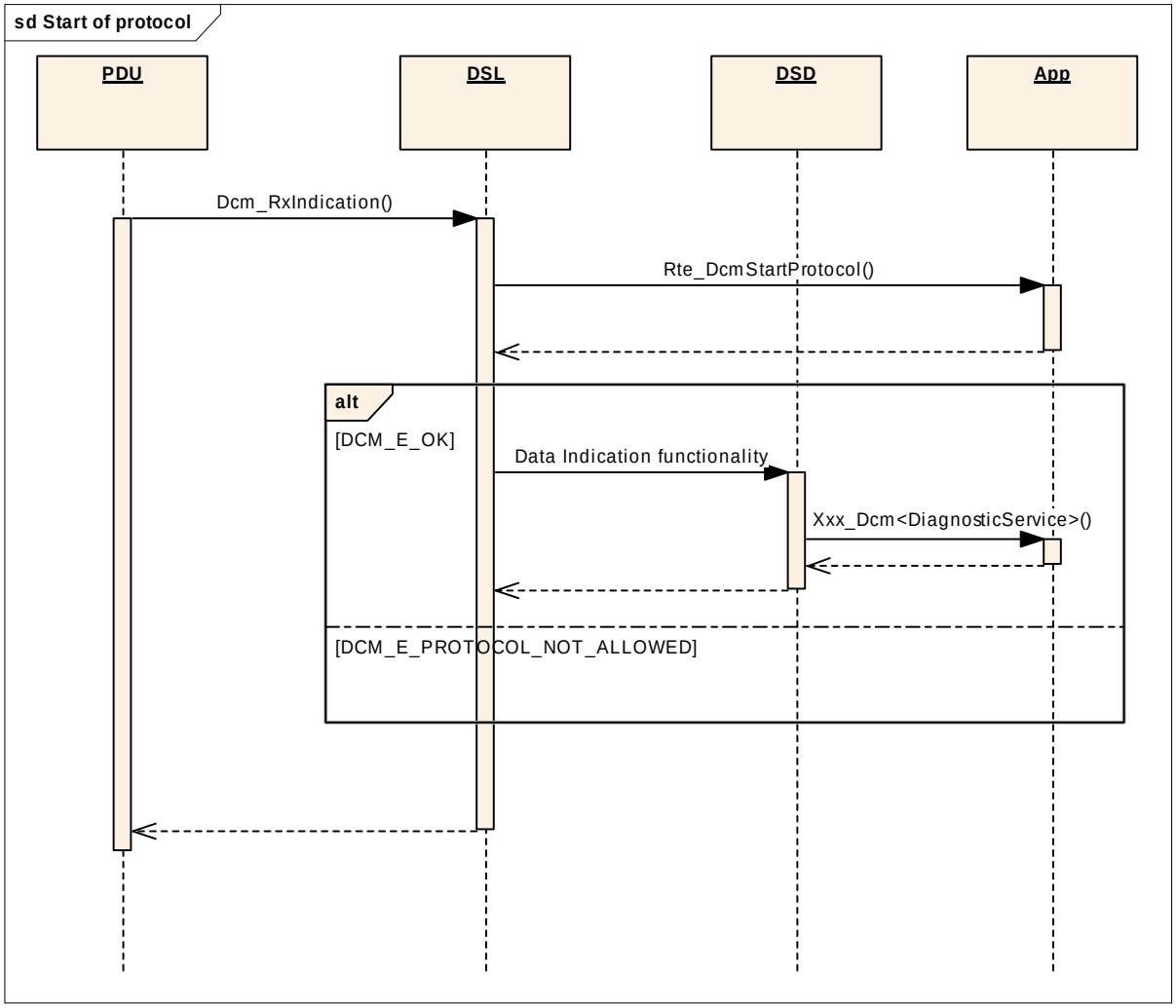
As well, after a **PduR_DcmTransmit()** request from DCM to Pdu-Router, data exchanges with **Dcm_ProvideTxBuffer()** service, are not show on the sequence diagrams.

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

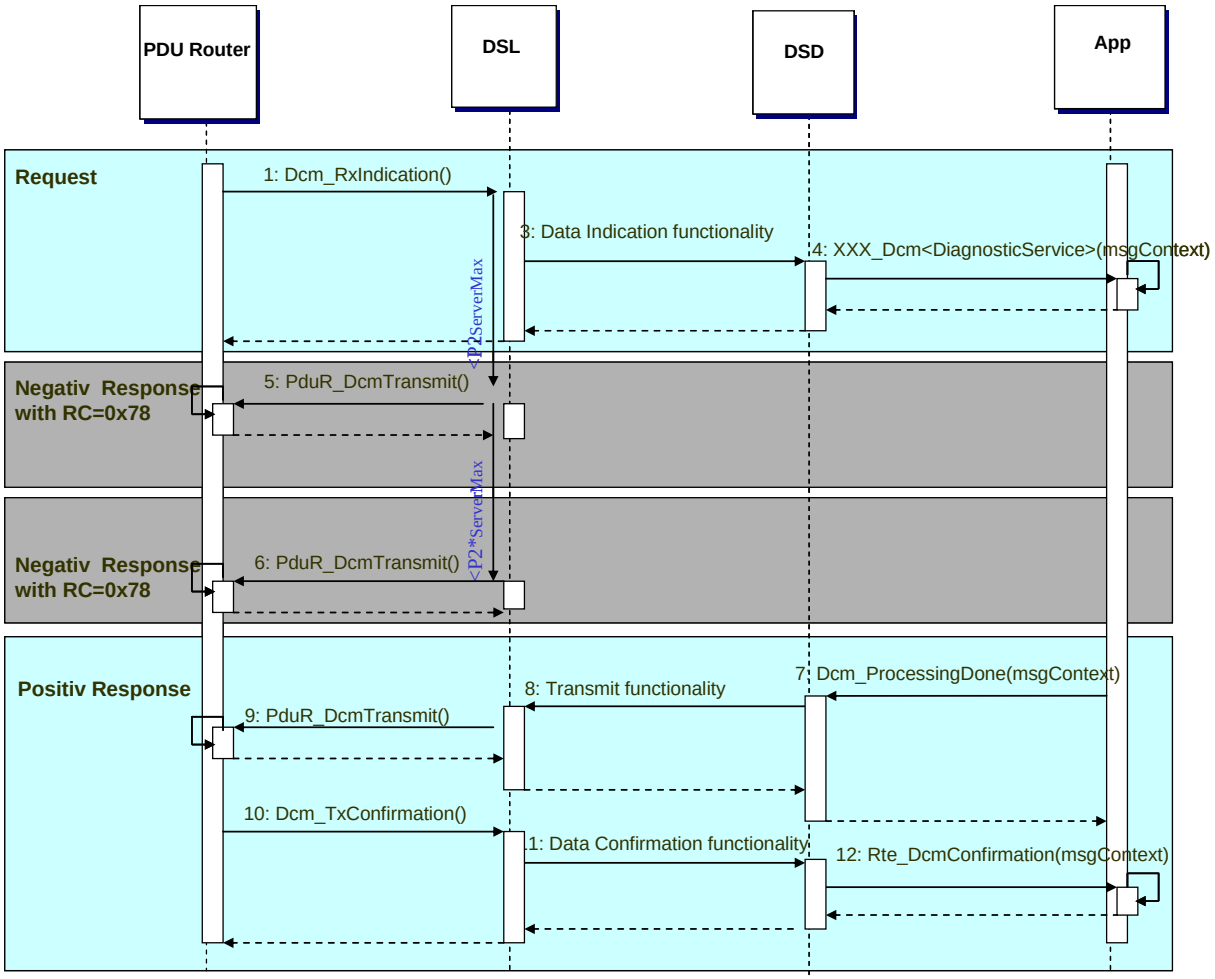
In all sequence charts for "App" and "RTE" the Central Diagnostic Software component is meant.

9.2 DSL (Diagnostic Session Layer)

9.2.1 Start Protocol

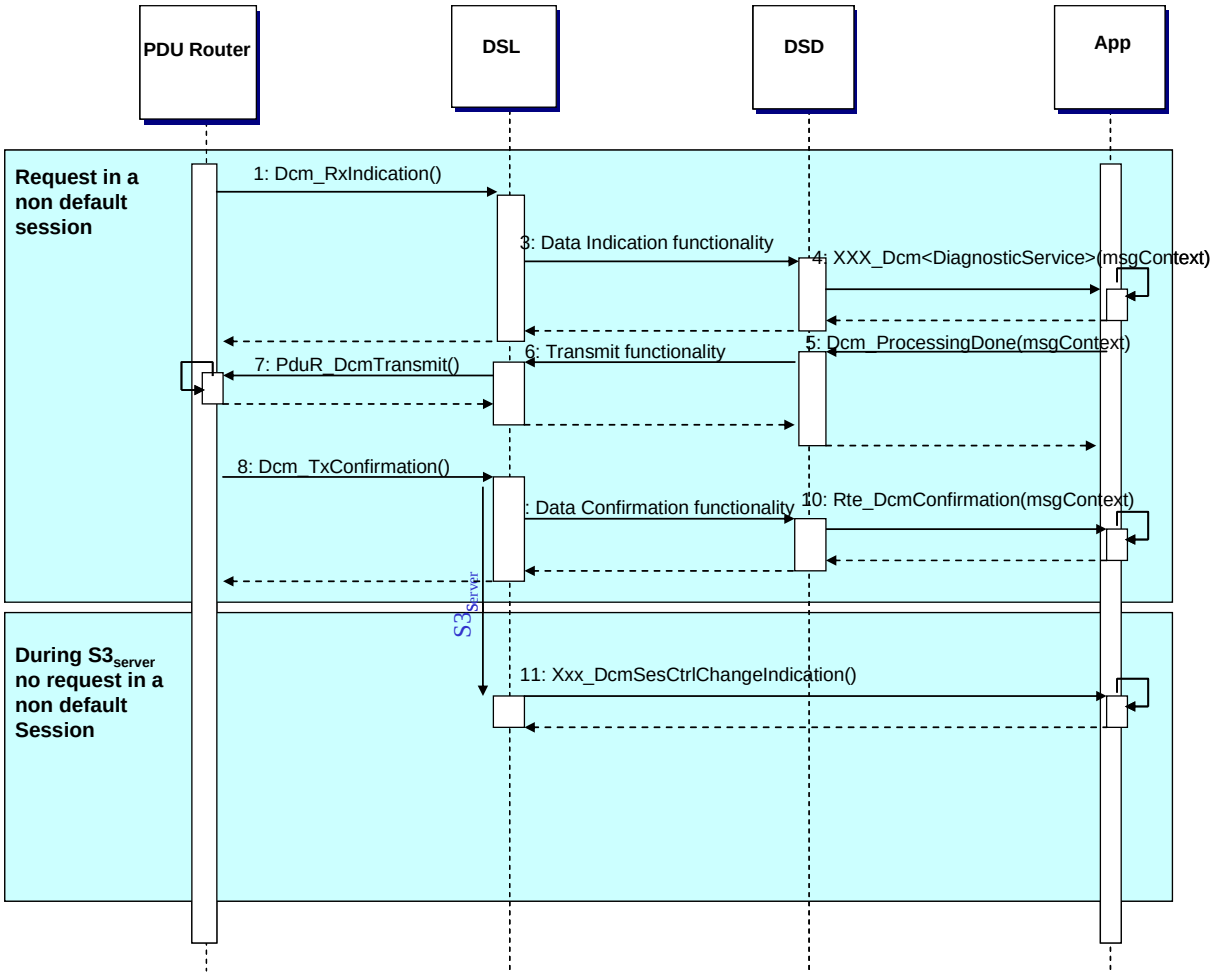


9.2.2 Process Busy behavior



Internally DSL calculates the time to response the tester. In case if the upper Layer (e.g. Application) 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.

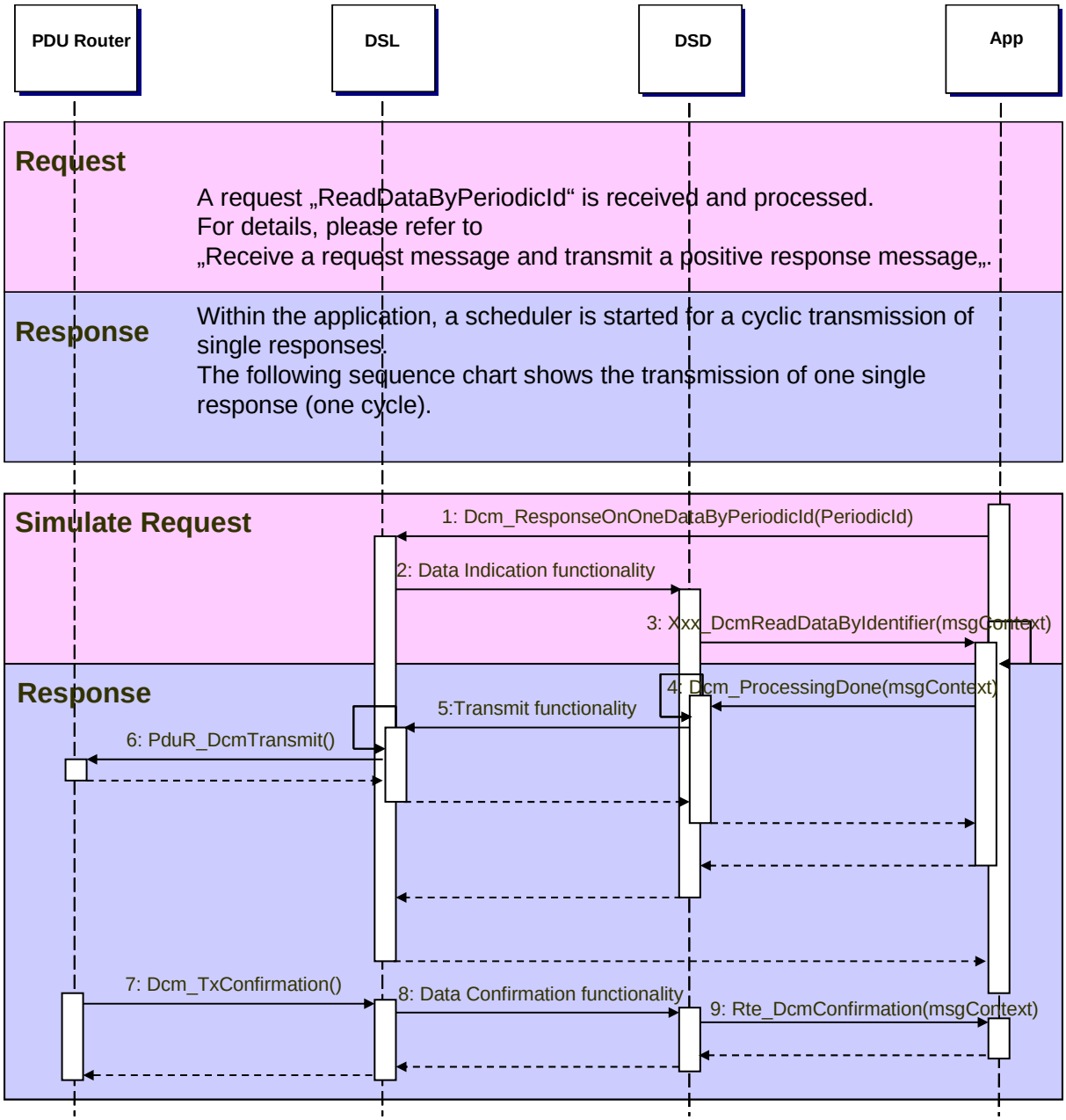
9.2.3 Update Diagnostic Session Control when timeout occurs



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

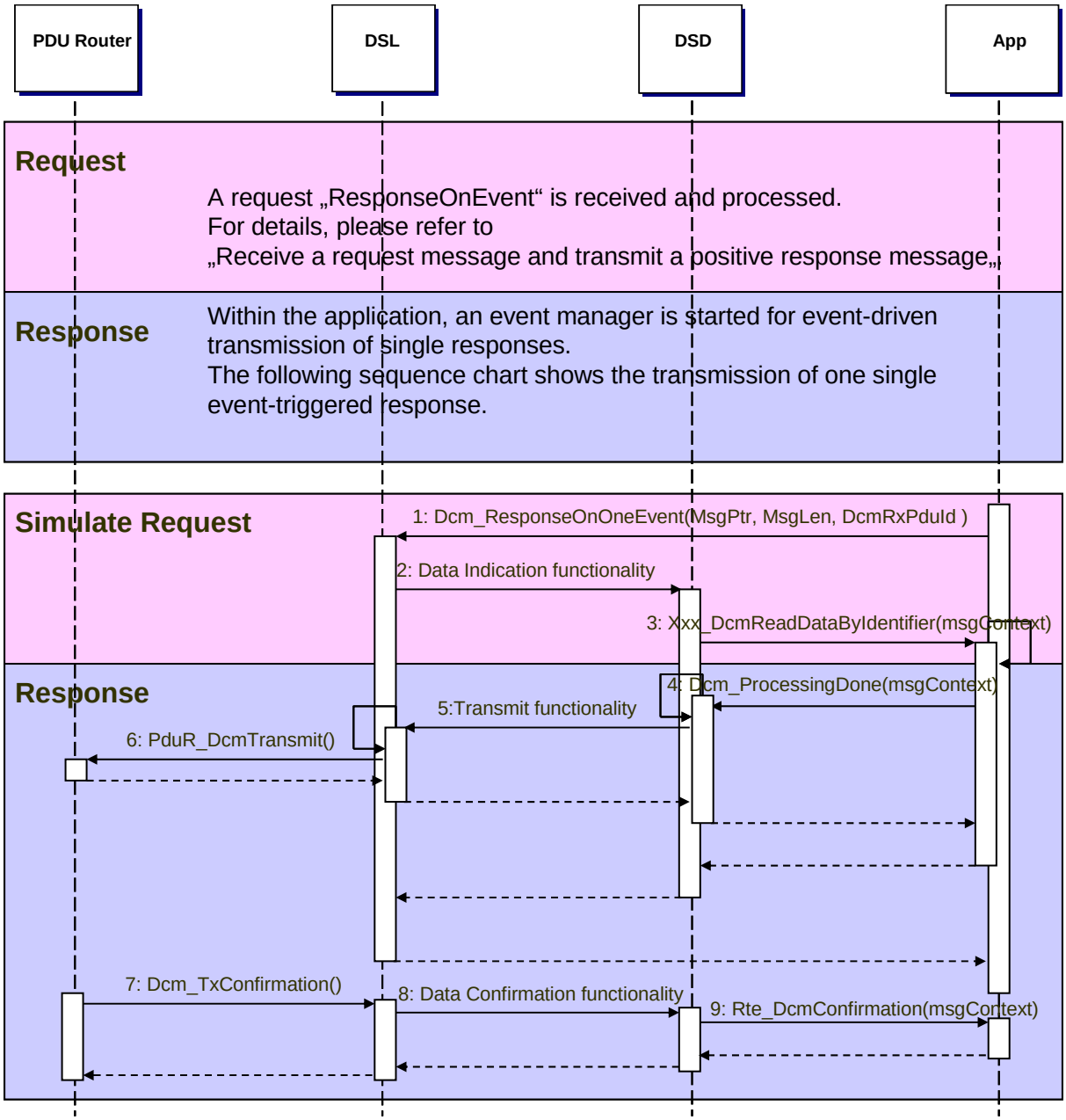
S3server timeout timer will be started with every data confirmation from the PDU Router.

9.2.4 Process single response of ReadDataByPeriodicIdentifier



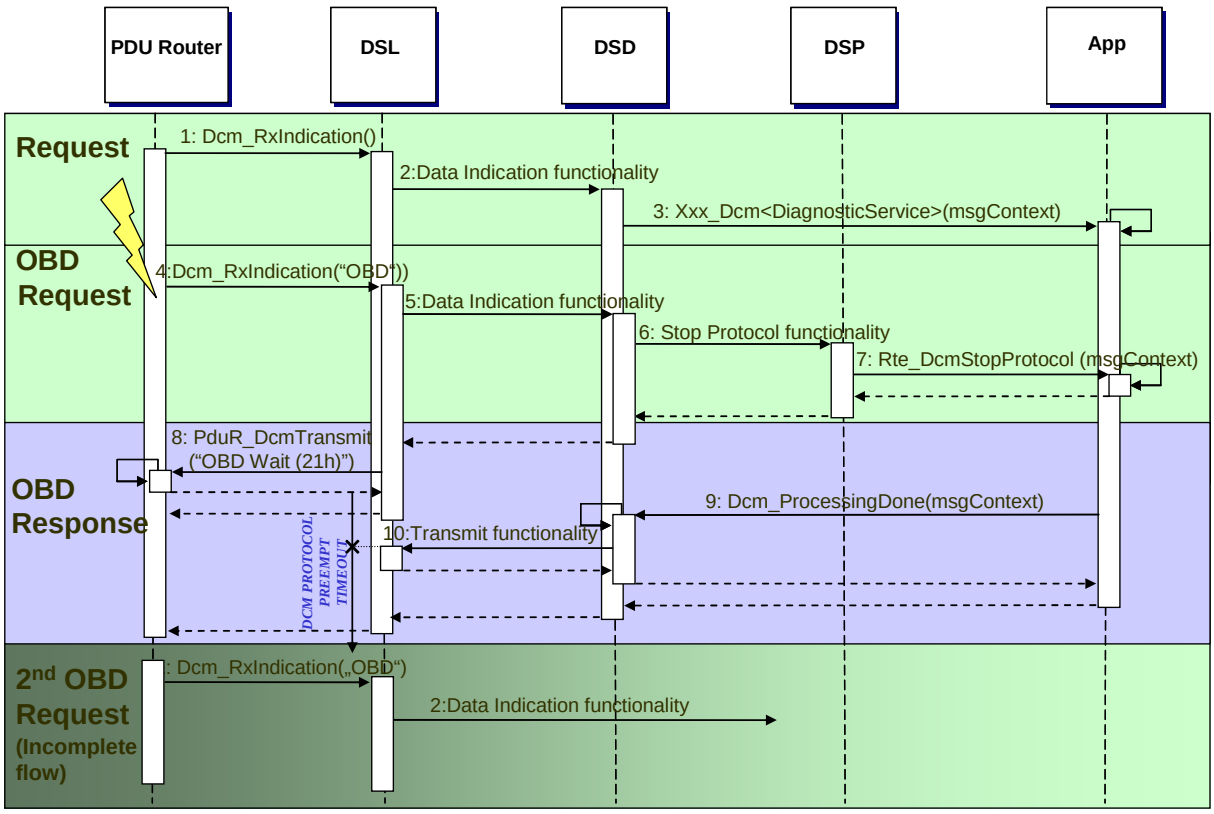
Application requests sampling and transmission of Periodic Identifier data, when an event to Periodic Identifier occurs (i. e. a given time period is over). The application initiates the sending of one periodic identifier calling the function **Dcm_ResponseOneDataByPeriodicId()** provided by the DSL.
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 [6]) and the low byte is set to value of the PeriodicId.
DCM is not able to receive another Periodic Identifier event request from application, 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 Application 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 or application, depending on the configuration given by the Service Identifier table.
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 ***DCM_PROTOCOL_PRIO***).

The following is processed on reception of 1st OBD request:
- Application is requested to stop already started processing (done with **RTE_DcmStopProtocol()**) and to reset to a stable state (e.g. switch of digital IOs,..).
- DSL responses with a negative response "BusyRepeatRequest" (NRC 0x21) to OBD tester.
- Timeout tracking of application finishes is started (timeout value configured in parameter ***DCM_PROTOCOL_PREEMPT_TIMEOUT*** of the preempting protocol (here OBD protocol))

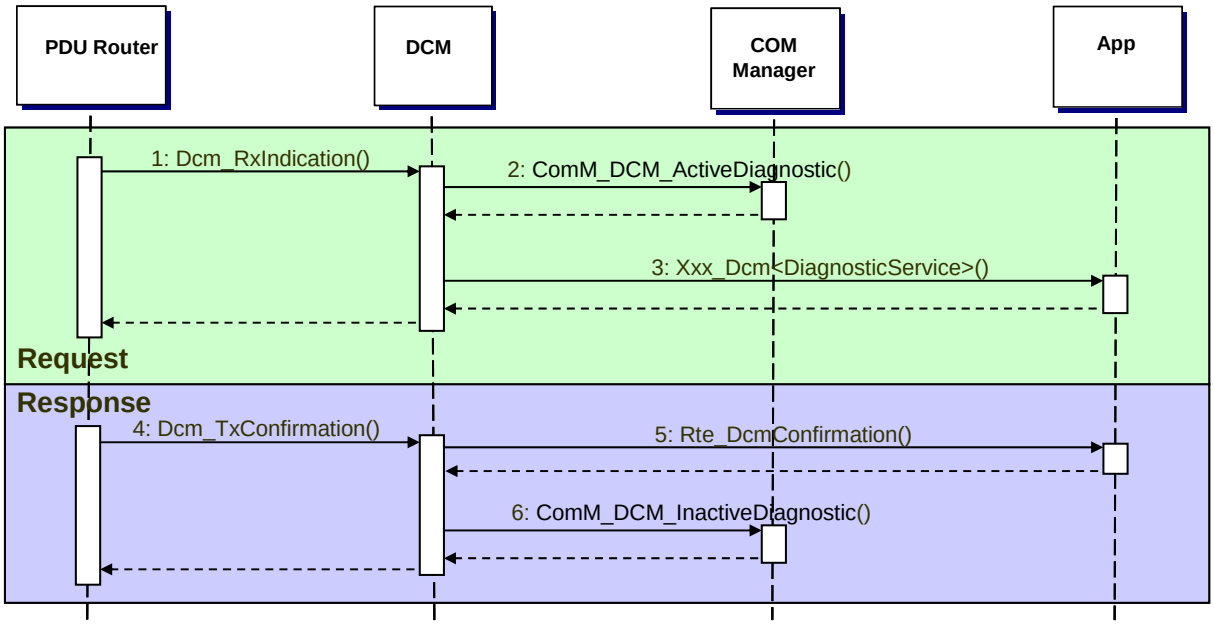
As long application is not finished (finish is indicated with **Dcm_ProcessingDone()**) or no timeout occurs, DSL responses with negative response "BusyRepeatRequest".

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.

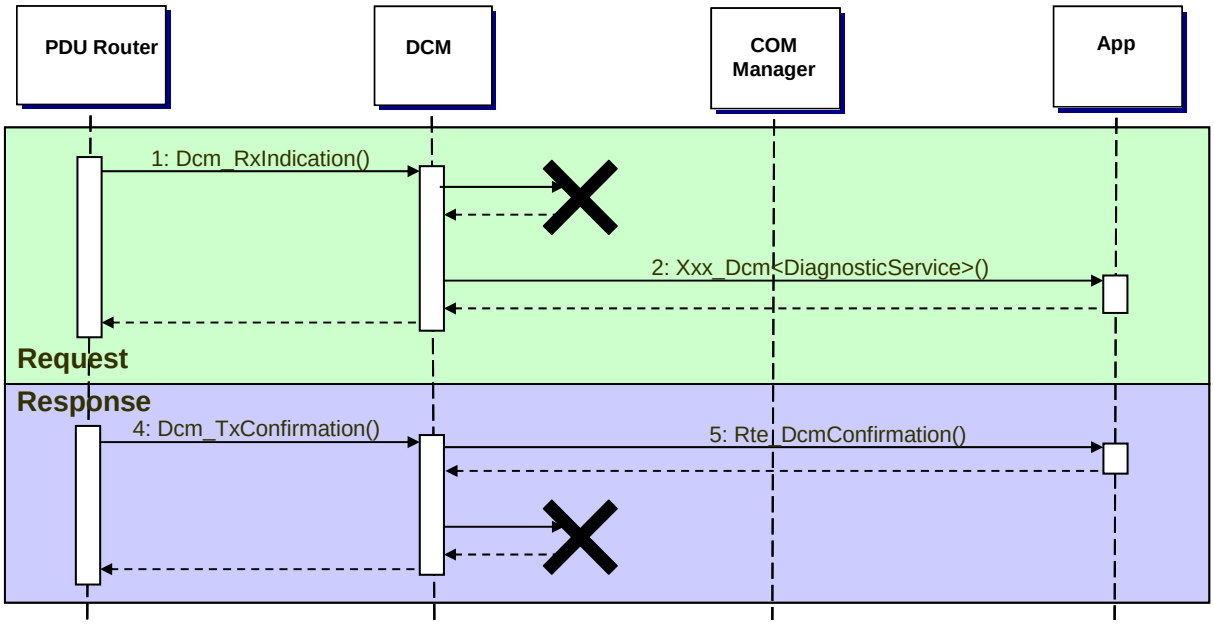
If application triggers no **Dcm_ProcessingDone()**, DSL runs into timeout and switches directly to further processing of preempting protocol.

9.2.7 Interface to ComManager

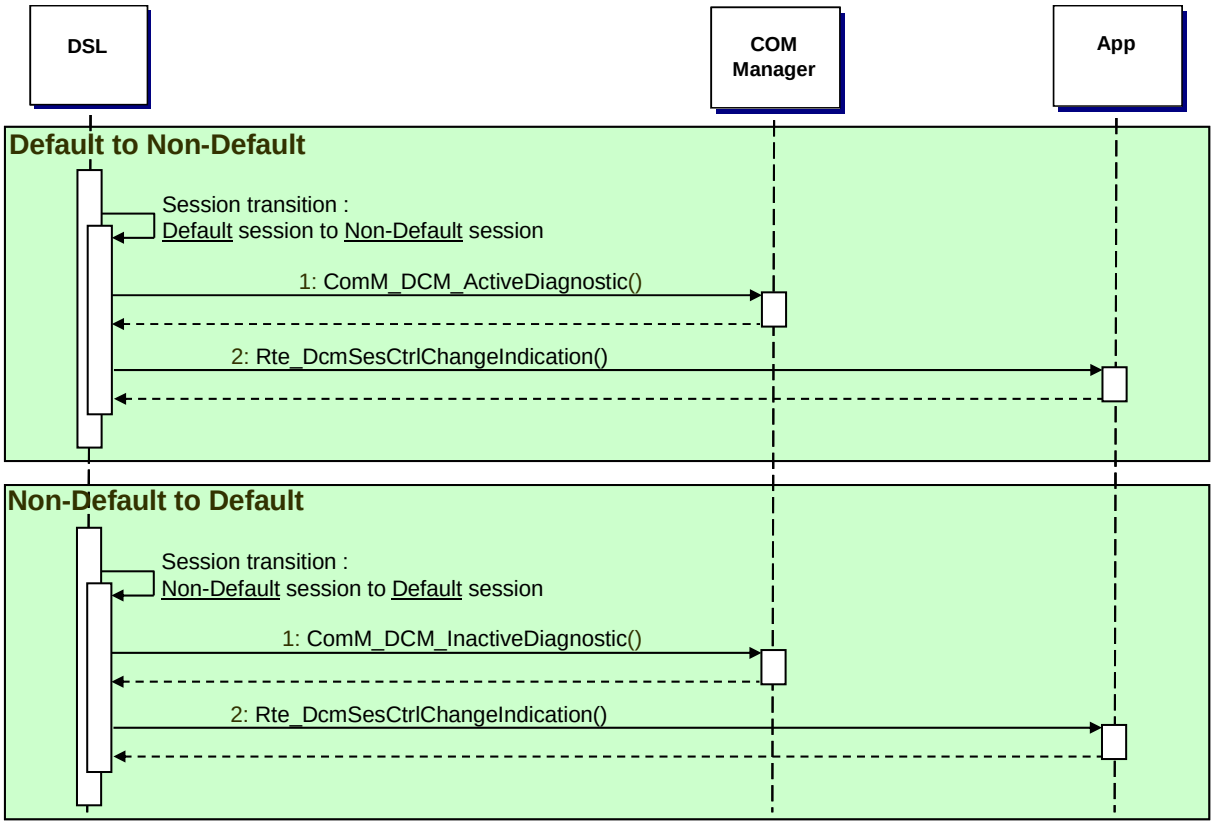
9.2.7.1 Handling in Default Session



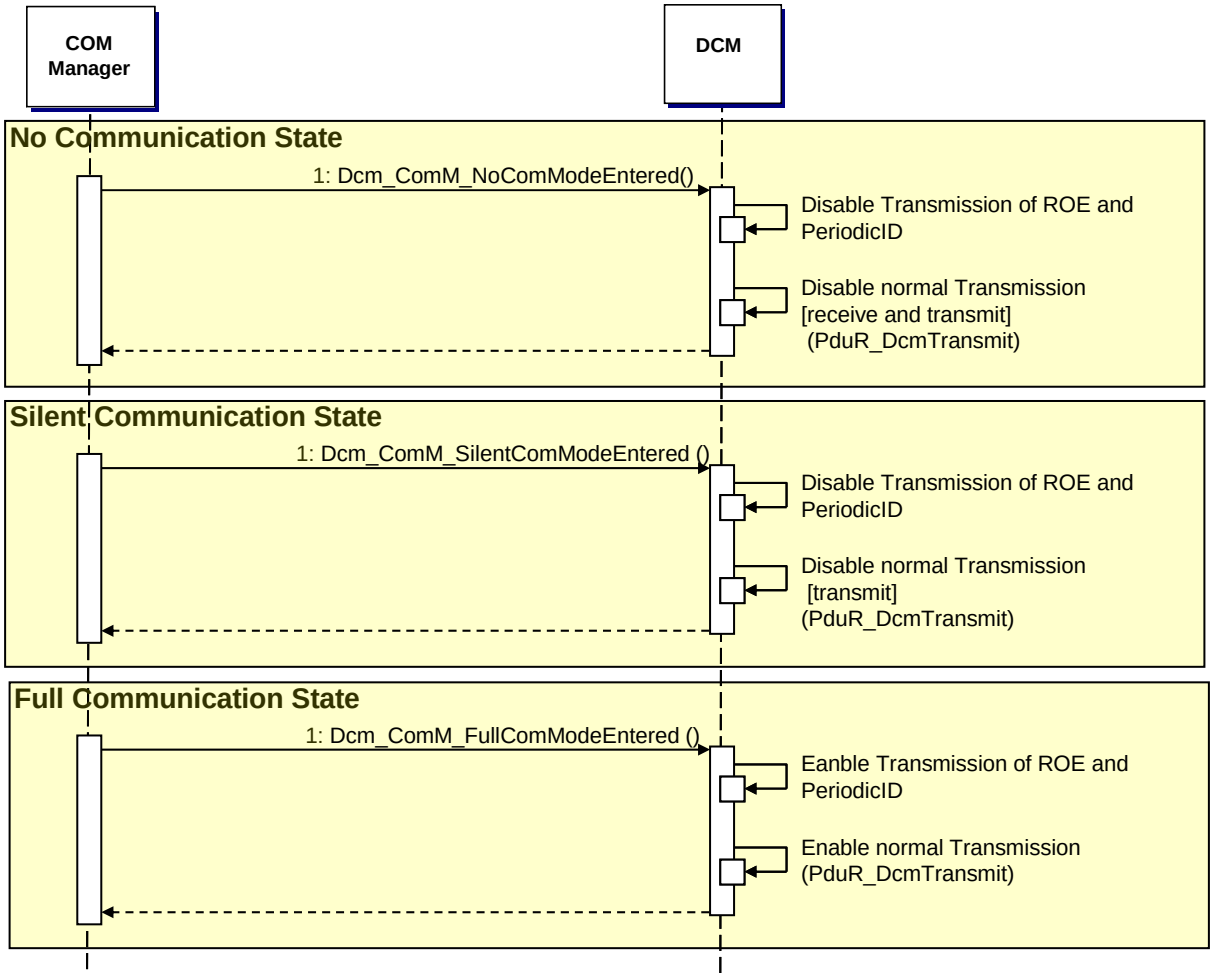
9.2.7.2 Handling in Non-Default Session



9.2.7.3 Session transitions

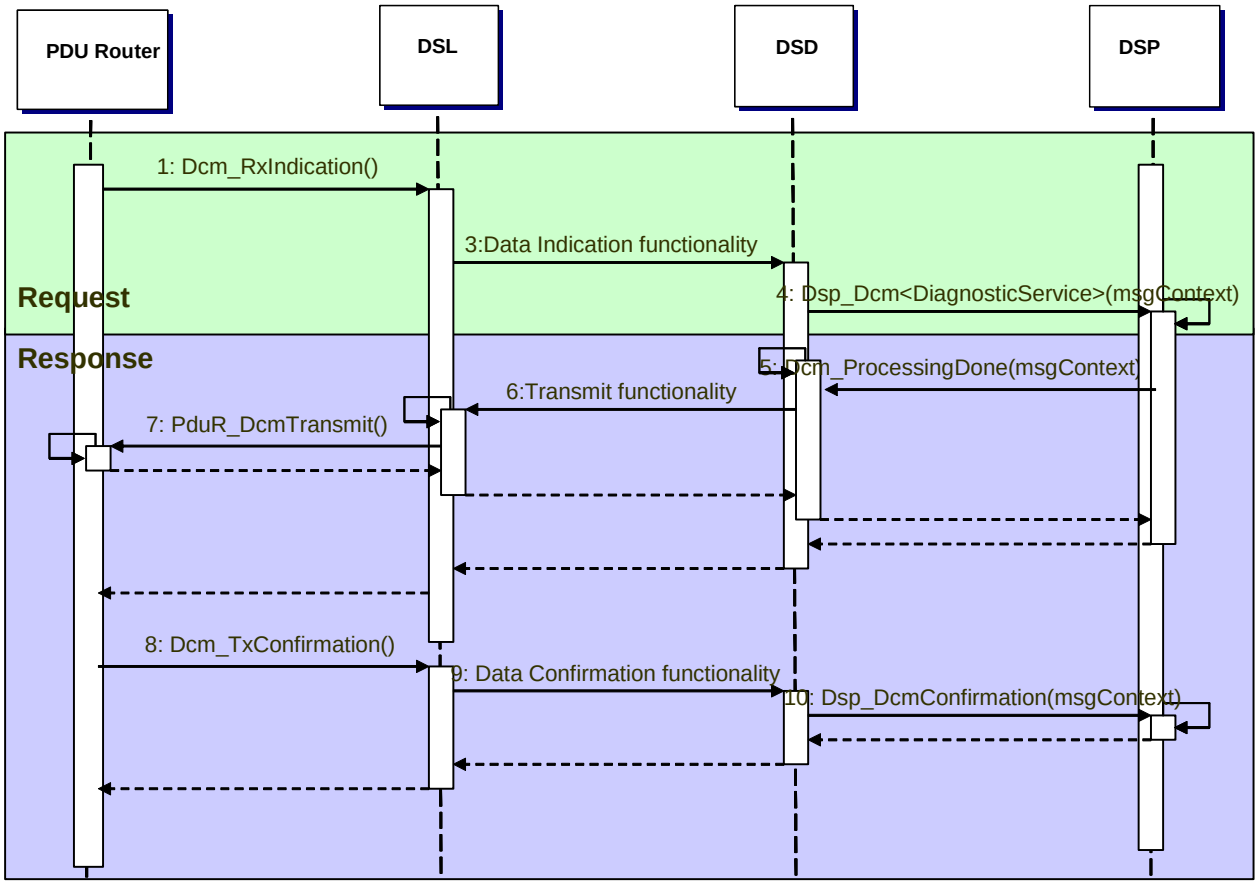


9.2.7.4 Communication States

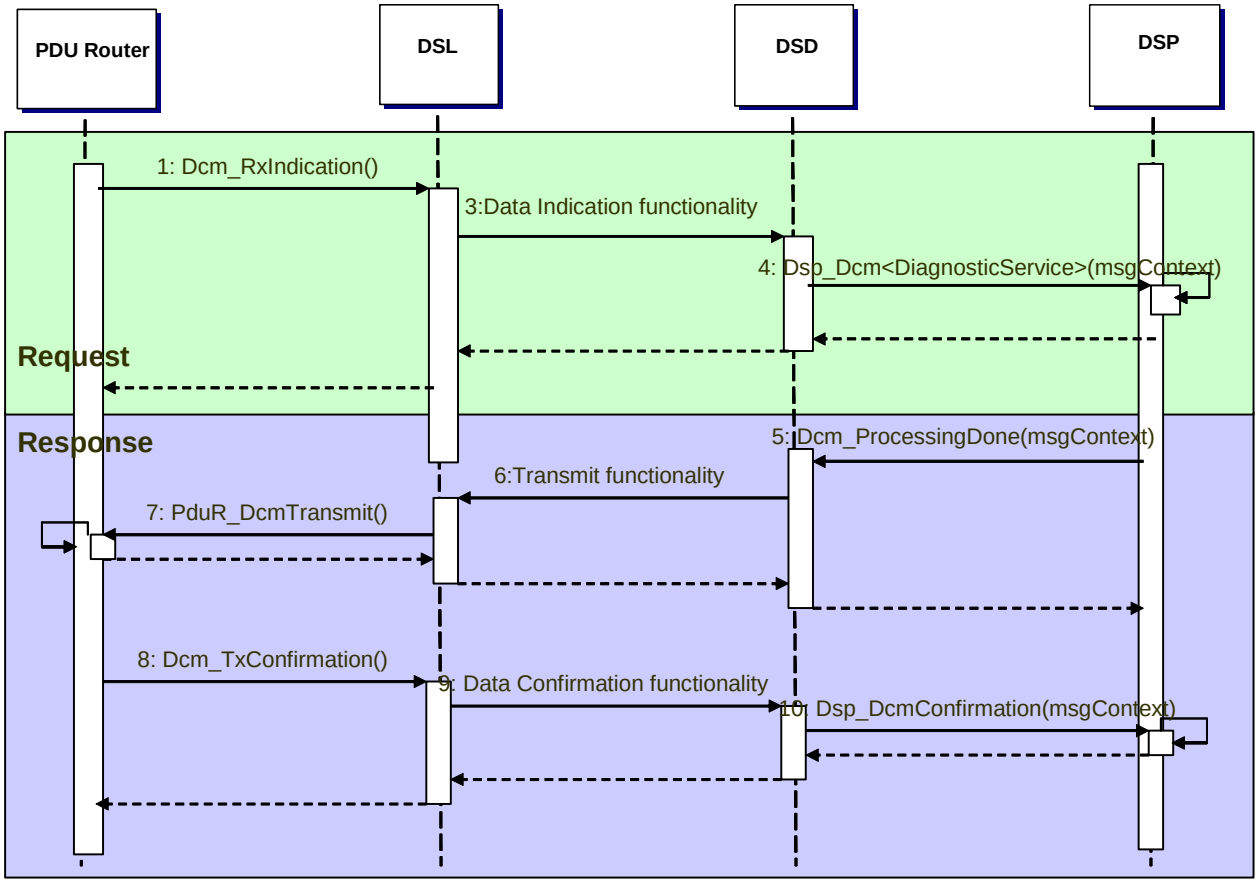


9.3 DSD (Diagnostic Service Dispatcher)

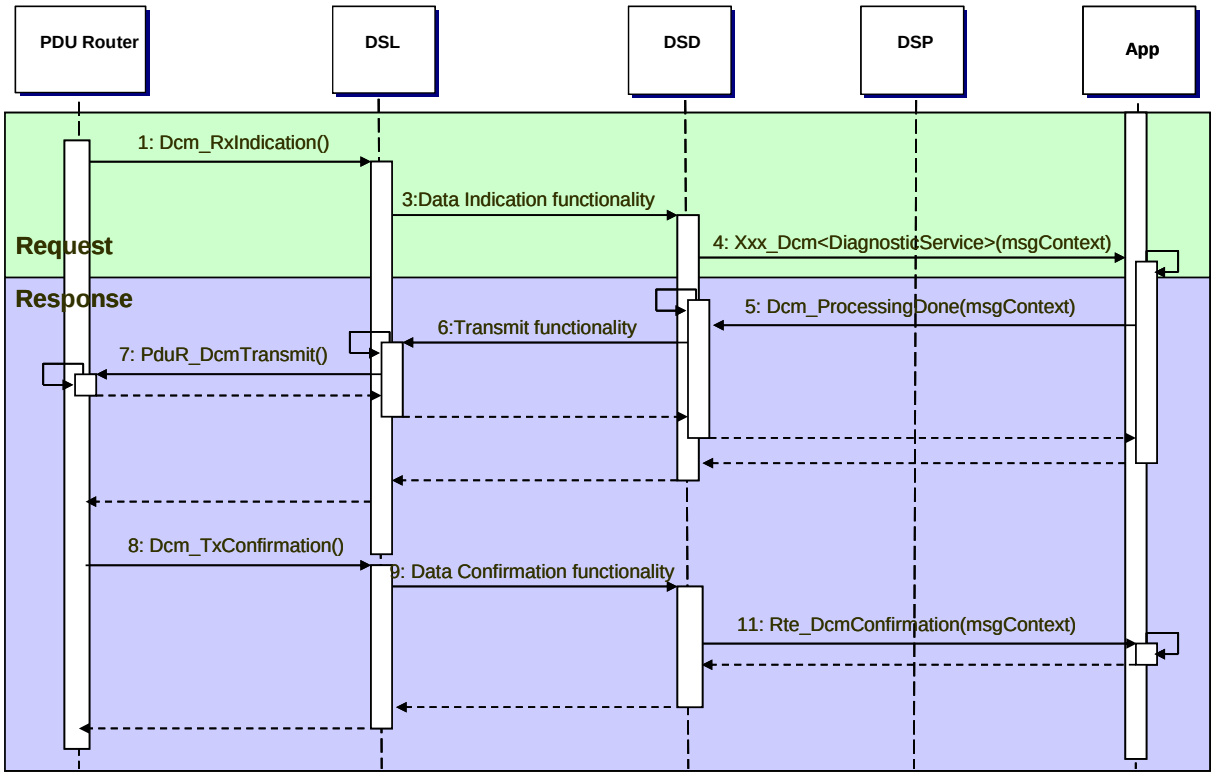
9.3.1 Receive a request message and transmit a positive response message – synchronous transmission



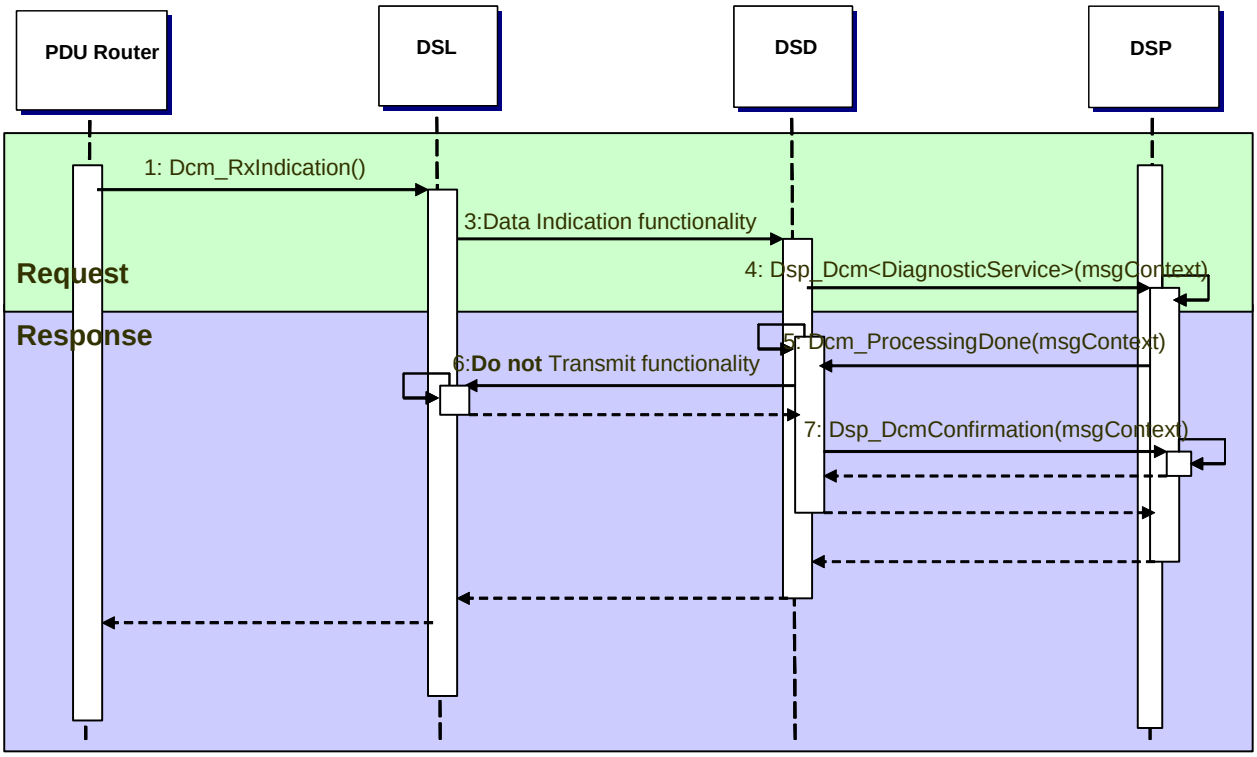
9.3.2 Receive a request message and transmit a positive response message – asynchronous transmission



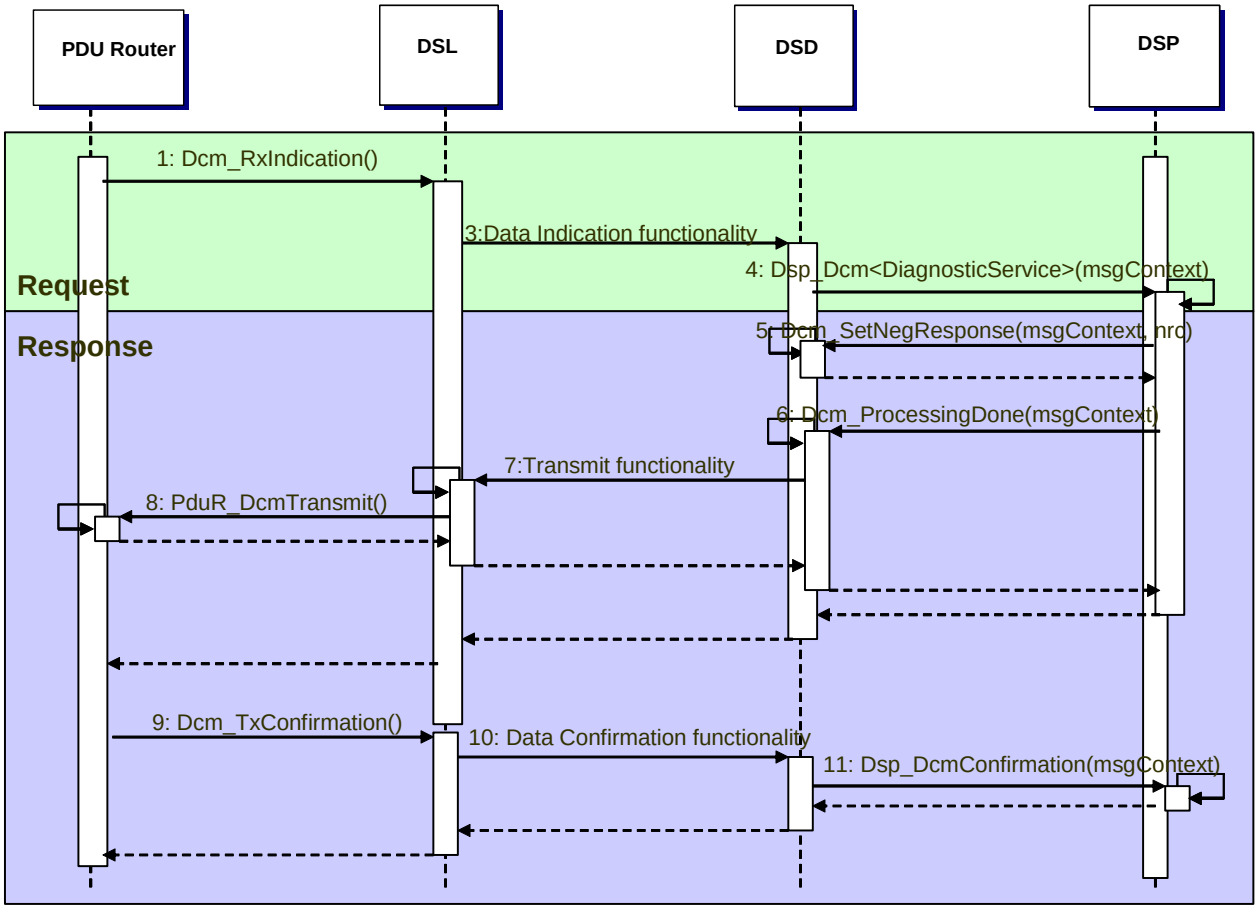
9.3.3 Receive a request message and transmits a positive response – synchronous transmission by application



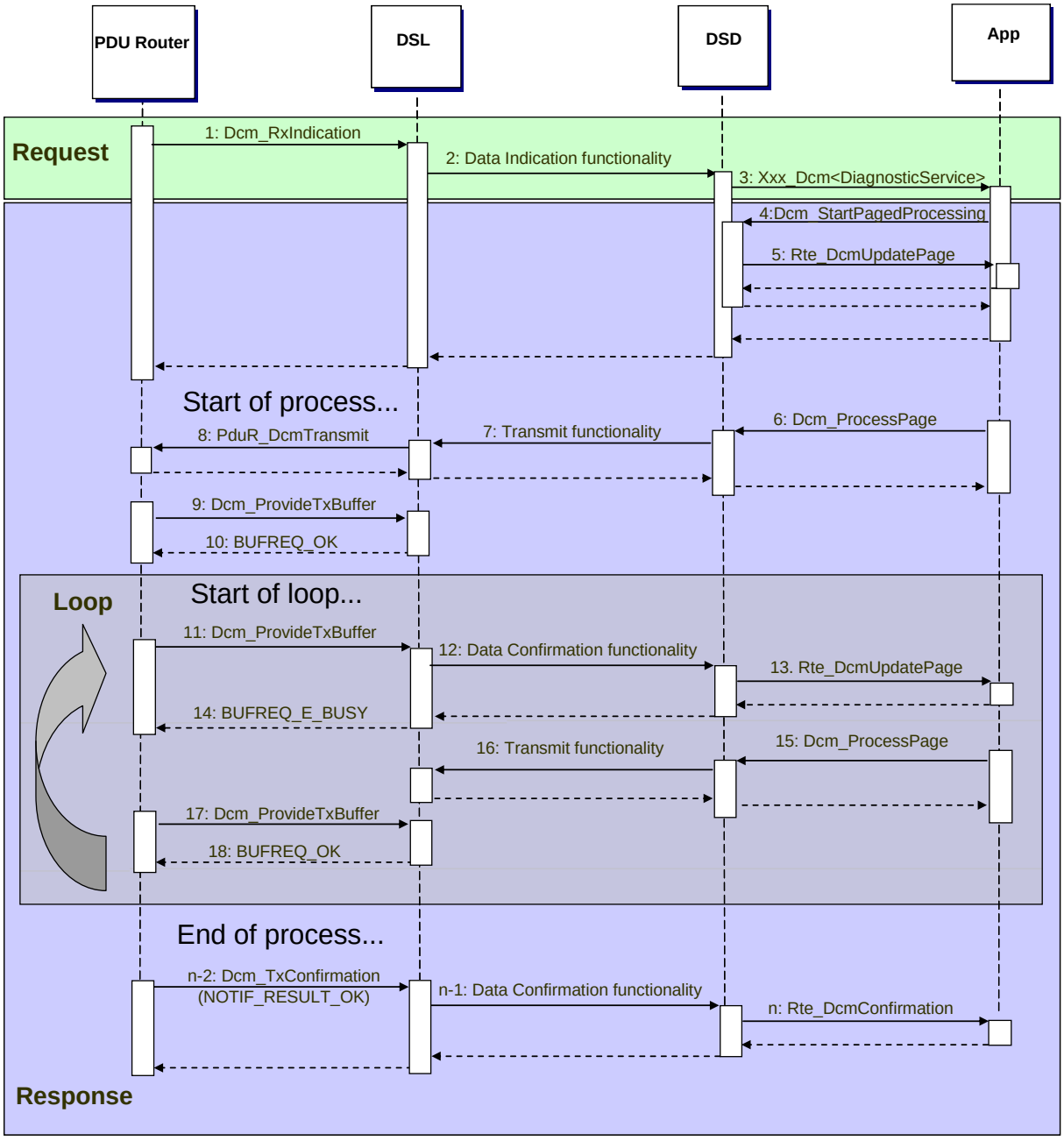
9.3.4 Receive a request message and suppression a positive response



9.3.5 Receive request message and transmit negative response message



9.3.6 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 application gives the complete response length to DCM and starts PagedBuffer handling. This API starts no transmission!

5) RTE_DcmUpdatePage()
DCM requests data to be transmitted.

6) Dcm_ProcessPage

With this API, the application 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 application.

12 + 13) RTE_DcmUpdatePage
DCM requests the application to fill the next page

14) By returning “BUFREQ_E_BUSY” DCM indicates that the buffer has to be filled from the application.

15) Dcm_ProcessPage()
With this API, the application 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 application

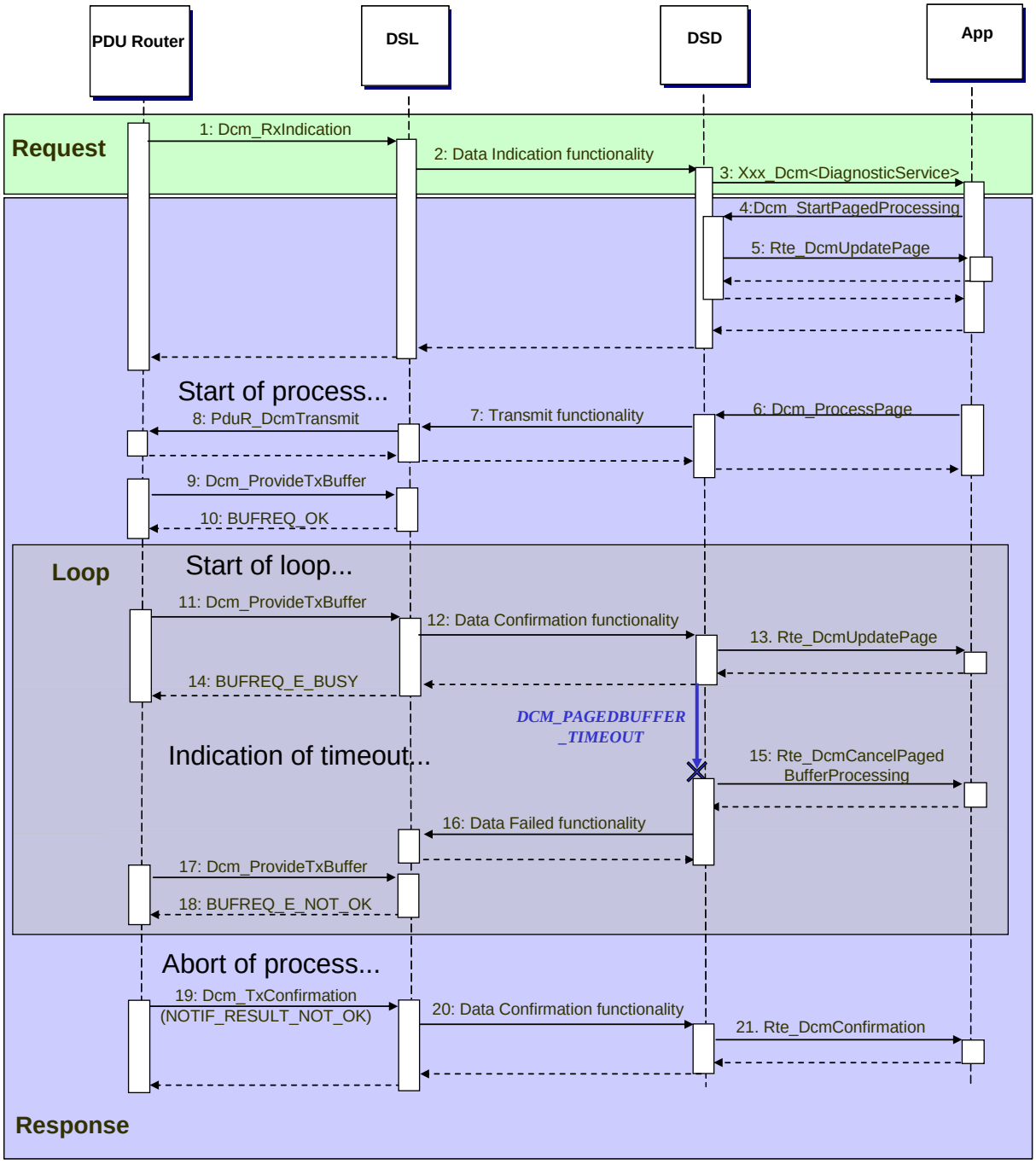
The APIs 4, 5 and 6 are needed only for PagedBuffer transmission

Page buffer timeout handling:

DCM react in the following described way, when application 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 application (13: RTE_DcmUpdatePage()), DCM start a timeout supervision. If timeout (value configured in DCM_PAGEDBUFFER_TIMEOUT) occurs, before application requests next page (Dcm_ProcessPage()) following error handling is carried out in DCM:

- DCM stops further processing of paged buffer (item 15)
- DCM requests application (14: RTE_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).


PDU Router
DSL
DSD
App
Request
1: Dcm_RxIndication
2: Data Indication functionality
3: Xxx_Dcm<DiagnosticService>
4:Dcm_StartPagedProcessing
5: Rte_DcmUpdatePage
Start of process...
8: PduR_DcmTransmit
7: Transmit functionality
6: Dcm_ProcessPage
9: Dcm_ProvideTxBuffer
10: BUFREQ_OK
Loop
Start of loop...
11: Dcm_ProvideTxBuffer
12: Data Confirmation functionality
13. Rte_DcmUpdatePage
14: BUFREQ_E_BUSY
DCM_PAGEDBUFFER_TIMEOUT
Indication of timeout...
15: Rte_DcmCancelPaged BufferProcessing
16: Data Failed functionality
17: Dcm_ProvideTxBuffer
18: BUFREQ_E_NOT_OK
Abort of process...
19: Dcm_TxConfirmation (NOTIF_RESULT_NOT_OK)
20: Data Confirmation functionality
21. Rte_DcmConfirmation
Response

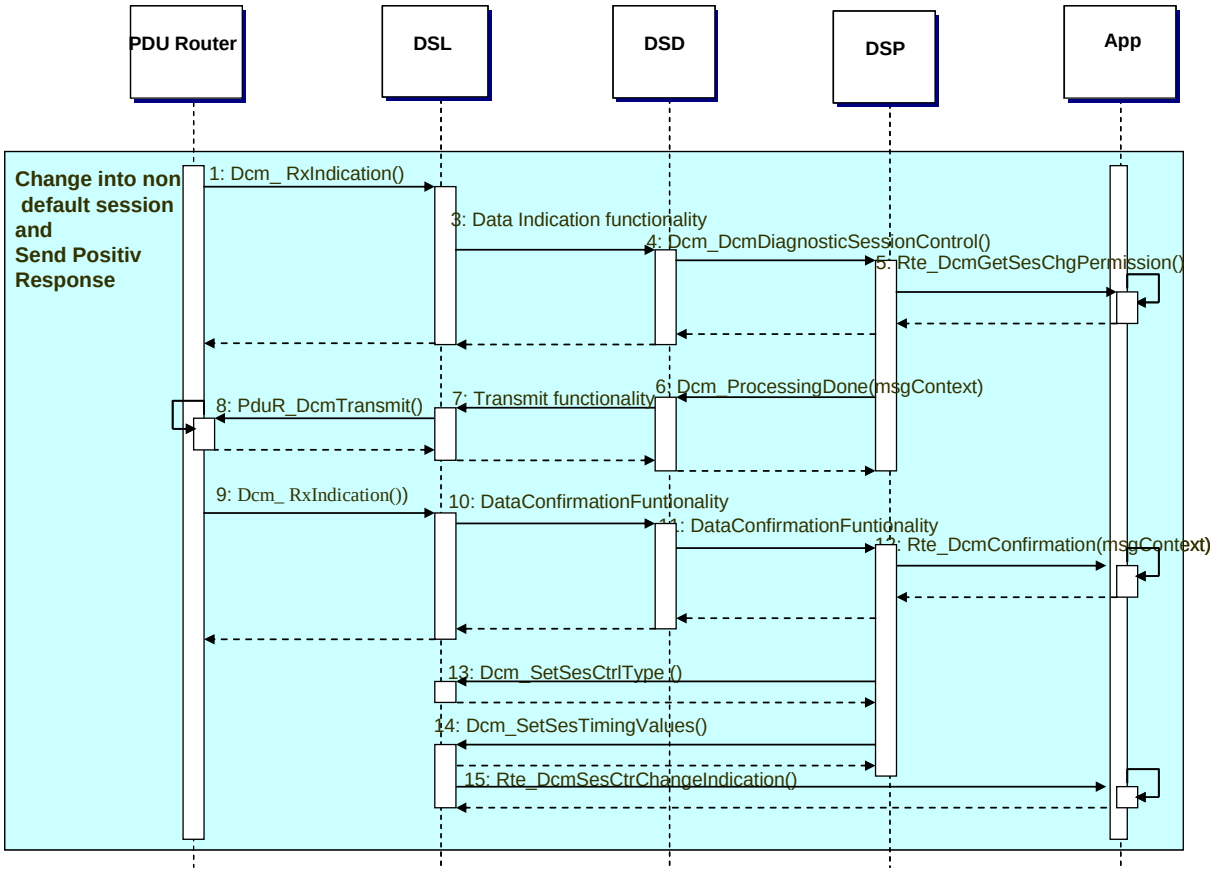
9.4 DSP (Diagnostic Service Processing)

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

Please refer to chapter 9. in [6].

9.4.2 Interface special services

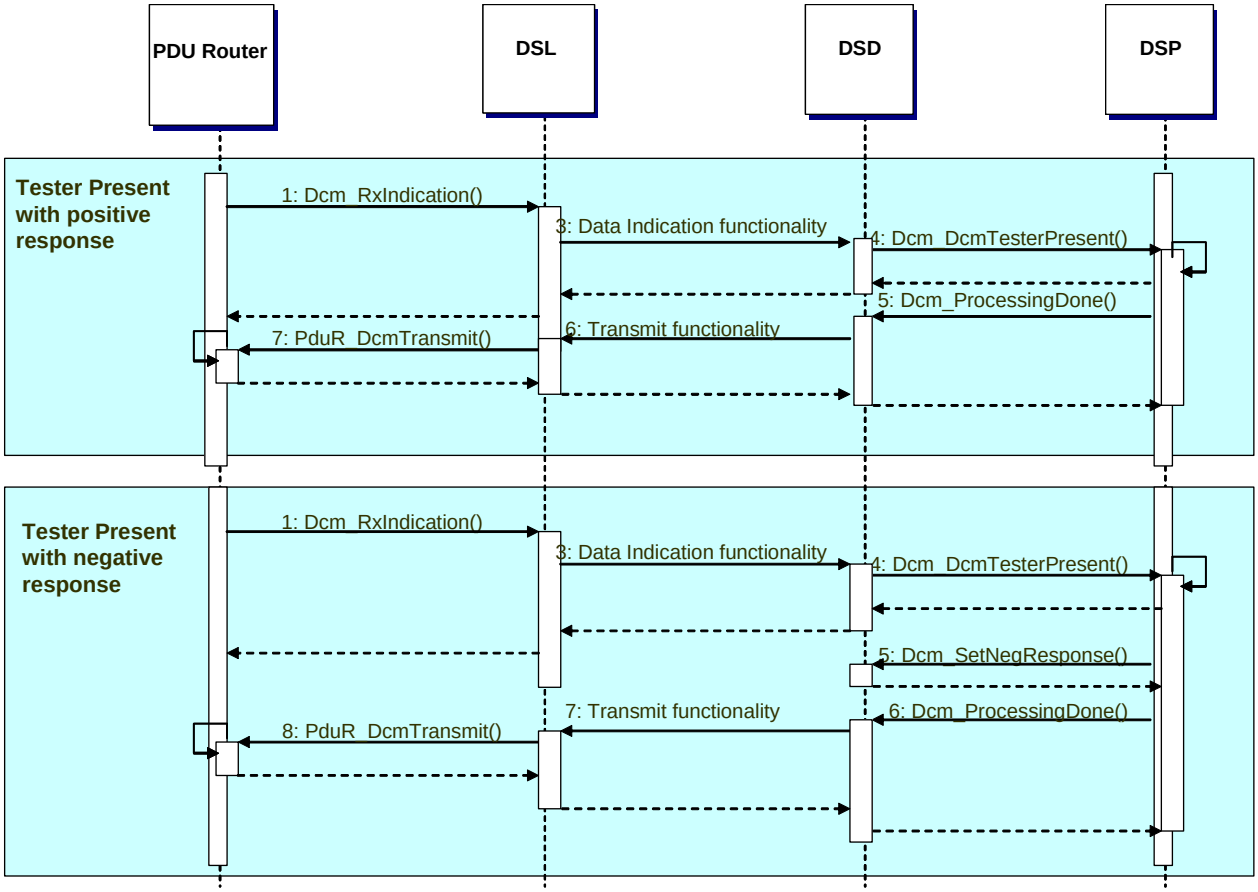
9.4.2.1 Process Diagnostic Session Control



On a Diagnostic Session Control request from tester DSP / application 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 (**RTE_DcmSesCtrlChangeIndication()**).

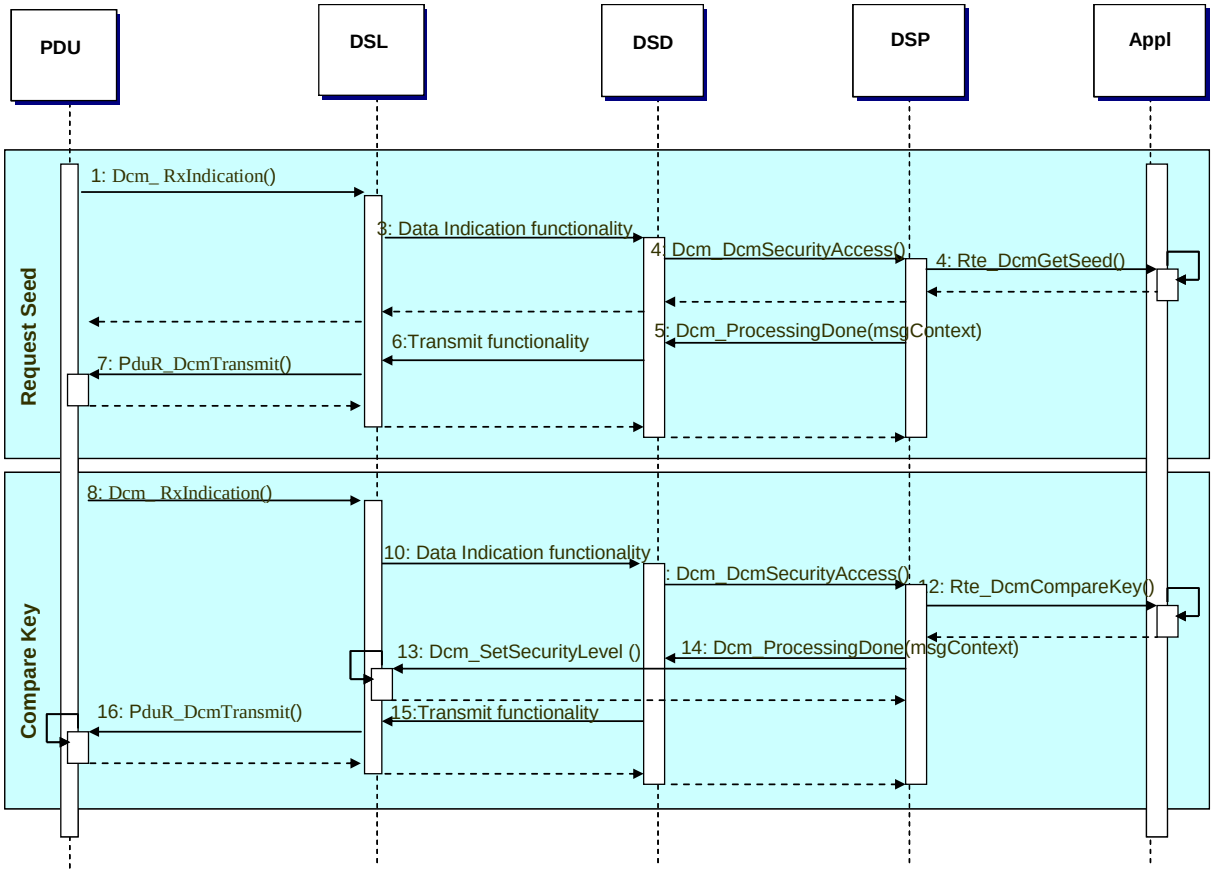
9.4.2.2 Process Tester Present



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, 0, 10.4, 10.5 specifies the structure (containers) and the parameters of the module DCM.

Chapter 0 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 *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

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 *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

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 *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* 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 *Post Build*.

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 complile 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 Configurable parameters

10.2.1.1 COMPONENT_WIDE_PARAMETERS

SWS Item	[Dcm075:] [Dcm076:]
Container Name	COMPONENT_WIDE_PARAMETERS
Description	This container contains the configuration (parameters) for Component wide parameters
Configuration Parameters	

Name	DCM_REQUEST_INDICATION_ENABLED		
Description	Allow to enable or disable the request indication mechanism		
Type	#define		
Unit	--		
Range	CFG_ON		Request Indication handling enabled
	CFG_OFF		Request Indication handling disabled
Configuration Class	***Pre-compile***	x	Mandatory for all Variant
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	DCM_DEV_ERROR_DETECT		
Description	Preprocessor switch to enable or disable the Development Error Detection (DET) mechanism.		
Type	#define		
Unit	--		
Range	CFG_ON		Development Error Detection (DET) is enabled
	CFG_OFF		Development Error Detection (DET) is disabled
Configuration Class	***Pre-compile***	x	Mandatory for all Variant
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	None		

Name	DCM_VERSION_INFO_API		
Description	Preprocessor switch to enable or disable the output Version info of the functionality.		
Type	#define		
Unit	--		
Range	CFG_ON		Version info is enabled
	CFG_OFF		Version info is disabled
	Pre-compile	x	Mandatory for all Variant
	Link time	--	--

	Post Build	--	--
Scope	ECU		
Dependency	None		

Name	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.		
Type	uint8		
Unit	ms		
Range	0		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	ECU		
Dependency	None		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.2.1.2 PAGE_BUFFER_CFG

SWS Item	[Dcm068]
Container Name	PAGE_BUFFER_CFG
Description	This container contains the configuration (parameters) for Page Buffer handling
Configuration Parameters	

Name	DCM_PAGEDBUFFER_ENABLED		
Description	Allow to enable or disable the Page buffer mechanism.		
Type	#define		
Unit	--		
Range	CFG_ON		Page Buffer handling enabled
	CFG_OFF		Page Buffer handling disabled
Configuration Class	***Pre-compile***	x	Mandatory for all Variant
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	DCM_PAGEDBUFFER_TIMEOUT		
Description	Allow to configure the Timeout (in ms) towards the application for filling the next page.		
Type	uint16		
Unit	ms		
Range	--		--
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	--	--

Scope	ECU
Dependency	This parameter is only relevant if the Page Buffer handling is enabled.

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.3 DSL (Diagnostic Session Layer)

10.3.1 Configurable parameters

10.3.1.1 DIAGNOSTIC_CONNECTION_TABLE

SWS Item	--
Container Name	DIAGNOSTIC_CONNECTION_TABLE
Description	This container contains links between DIAGNOSTIC_PROTOCOL_TABLE and the included containers SERVICE_IDENTIFIER_TABLE / DIAGNOSTIC_PROTOCOL_TX_TABLE / PROTOCOL_TIMING_STRUCTURE.DCM_TIMSTR_TYPE. Use Case: For more then one diagnostic testers (using different CAN cannels), it is necessary to specify multiple DCM_PROTOCOL_DCMTXPDUID per DCM_PROTOCOL_DCMRXPDUID
Configuration Parameters	

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DIAGNOSTIC_PROTOCOL_RX_TABLE	1	ECU	1..n (functional DCM_PROTOCOL_RX_ADDR_TYPE) / 1:1 (physical DCM_PROTOCOL_RX_ADDR_TYPE)
DIAGNOSTIC_PROTOCOL_TX_TABLE	1	ECU	1..n

10.3.1.2 DIAGNOSTIC_PROTOCOL_RX_TABLE

SWS Item	--		
Container Name	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 DCM_PROTOCOL_DCMRXPDUID can be given several times per protocol an that the combination from DCM_PROTOCOL_DCMRXPDUID and DCM_PROTOCOL_RX_ADDR_TYPE is unique		
Configuration Parameters			
Name	DCM_PROTOCOL_DCMRXPDUID		
Description	DcmRxPduId value for reception of requests / Multiple links shall be allowed. (e.g. one DcmRxPduId to receive func requests, one DcmRxPduId to receive phys requests)		
Type	uint8		
Unit	--		
Range	0...max	max = maxsize uint8	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_RX_ADDR_TYPE DIAGNOSTIC_PROTOCOL_TABLE.DCM_PROTOCOL_ID		

Name	DCM_PROTOCOL_RX_ADDR_TYPE		
Description	Declares the communication type of this DCM_PROTOCOL_DCMRXPDUID. PHYSICAL is used to 1 to 1 communication FUNCTIONAL is used at 1 to n communication		
Type	enumeration		
Unit	--		
Range	0	PHYSICAL	
	1	FUNCTIONAL	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_DCMRXPDUID DIAGNOSTIC_PROTOCOL_TABLE.DCM_PROTOCOL_ID		

Name	DCM_PROTOCOL_RX_BUFFER_ID		
Description	Link to buffer configuration (DCM_BUFFER_ID) for configuration of protocol Rx buffer For the Tx the larger Rx buffer is used It is allowed to share the buffer in Rx and Tx case.		
Type	uint8		
Unit	--		
Range	0...max	max = max numbers of DCM_BUFFER_ID - 1 (0 = first item)	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_DCMRXPDUID DIAGNOSTIC_BUFFER_CFG.DCM_BUFFER_ID		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DIAGNOSTIC_BUFFER_CFG	1	ECU	n..1

10.3.1.3 DIAGNOSTIC_PROTOCOL_TABLE

10.3.1.3.1 Row of DIAGNOSTIC_PROTOCOL_TABLE:

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

Name	DCM_PROTOCOL_ID		
Description	Name of the diagnostic protocol Type definition along Dcm_ProtocolType		
Type	Dcm_ProtocolType		
Unit	--		
Range	0..2	See Dcm_ProtocolType	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DIAGNOSTIC_PROTOCOL_RX_TABLE.DCM_PROTOCOL_DCMRXPDUID DIAGNOSTIC_PROTOCOL_RX_TABLE.DCM_PROTOCOL_RX_ADDR_TYPE		

Name	DCM_PROTOCOL_PREEMPT_TIMEOUT		
Description	Timeout (in ms) of preempting protocol until protocol needs to be started		
Type	uint16		
Unit	ms		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_ID		

Name	DCM_PROTOCOL_IDENTIFIER_TABLE_ID
Description	Link to the used diagnostic service table (SERVICE_IDENTIFIER_TABLE.DCM_SIDTAB_ID). One DCM_SIDTAB_ID per DCM_PROTOCOL_ID shall be used!
Type	uint8
Unit	--

Range	0..	0 = first item `max = max numbers of IDENTIFIER_TABLES - 1`	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_ID SERVICE_IDENTIFIER_TABLE.DCM_SIDTAB_ID		

Name	`DCM_PROTOCOL_PRIO`		
Description	Configuration of Protocol priority Used at protocol preemption handling		
Type	`uint8`		
Unit	--		
Range	0	protocol is not preemptable by other protocols	
	1,2,3,..	protocol is pre-emptable by other protocols which has lower values	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_ID		

Name	`DCM_PROTOCOL_TIME_TABLE_ID`		
Description	Link to the used diagnostic Time table (PROTOCOL_TIMING_STRUCTURE.DCM_TIMSTR_TYPE). Please notice that per protocol 2 Time tables shall be configured.		
Type	`Dcm_TimerModeType`		
Unit	--		
Range	DCM_DEFAULT	Configured Default timing parameters	
	DCM_LIMITS	Configured Timing parameters limits	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_ID PROTOCOL_TIMING_STRUCTURE.DCM_TIMSTR_TYPE		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DIAGNOSTIC_CONNECTION_TABLE	1	ECU	1..n
SERVICE_IDENTIFIER_TABLE	n	ECU	1..1
PROTOCOL_TIMING_STRUCTURE.DCM_TIMSTR_TYPE	2 (DCM_LIMITS table / DCM_DEFAULT table)	ECU	1..n

10.3.1.3.2 DIAGNOSTIC_PROTOCOL_TABLE (consists of rows):

SWS Item	--
Container Name	DIAGNOSTIC_PROTOCOL_TABLE
Description	This container contains the configuration (parameters) for the protocol configuration (for each protocol) This container contains Rows of DIAGNOSTIC_PROTOCOL_TABLE_ROW (needs to be configured per protocol)
Configuration Parameters	

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DIAGNOSTIC_PROTOCOL_TABLE_ROW	1..n	ECU	--

10.3.1.4 DIAGNOSTIC_PROTOCOL_TX_TABLE

SWS Item	--
Container Name	DIAGNOSTIC_PROTOCOL_TX_TABLE
Description	This container contains the configuration (parameters) for the protocol configuration of TX channel (for each protocol) The following parameters needs to be configured per protocol. Please keep in mind, that the parameter DCM_PROTOCOL_DCMTXPDUID can be given several times per protocol
Configuration Parameters	

Name	DCM_PROTOCOL_DCMTXPDUID		
Description	DcmTxPduId value for transmission of responses		
Type	uint8		
Unit	--		
Range	0...max	max = maxsize uint8	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_ID		

Name	DCM_PROTOCOL_TX_BUFFER_ID		
Description	Link to buffer configuration (DCM_BUFFER_ID) for configuration of protocol Tx buffer For the Tx the larger Rx buffer is used It is allowed to share the buffer in Rx and Tx case.		
Type	uint8		
Unit	--		
Range	0..max	max = max numbers of DCM_BUFFER_ID –1 (0 = first item)	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PROTOCOL_DCMRXPDUID DCM_BUFFER_ID		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DIAGNOSTIC_BUFFER_CFG	1	ECU	n..1

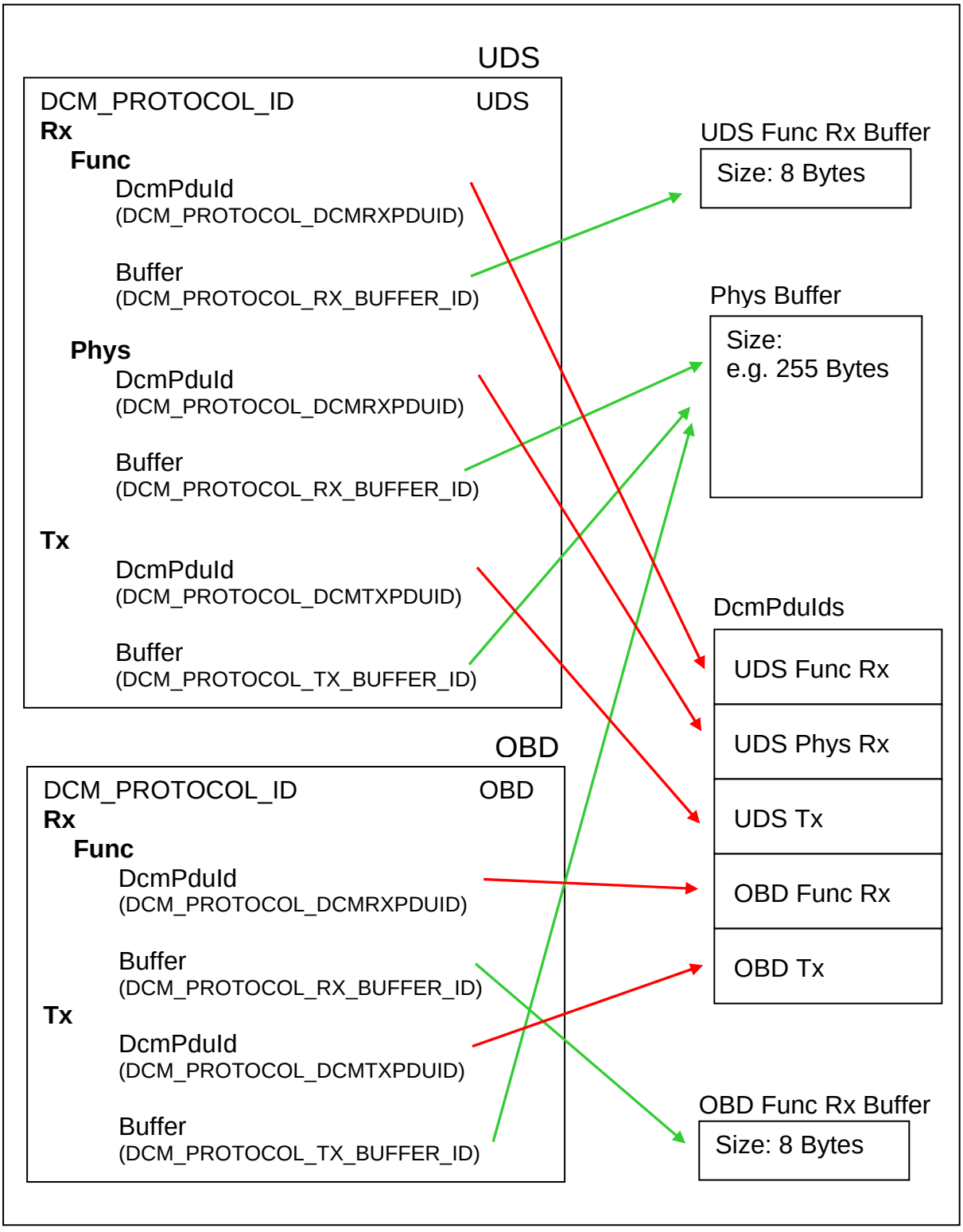


Figure 16 Examples of protocol configuration with focus on buffer / DcmPduId settings

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 DcmPduRxIds 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 DIAGNOSTIC_CONNECTION_TABLE and DIAGNOSTIC_PROTOCOL_RX_TABLE, DIAGNOSTIC_PROTOCOL_TX_TABLE and DIAGNOSTIC_PROTOCOL_TABLE is shown.
Please note:
The DIAGNOSTIC_PROTOCOL_RX_TABLE has two possible configurations:

- functional
- physical

The physical shall have a 1:1 (or 1:0) dependency to the DIAGNOSTIC_CONNECTION_TABLE.
(which means: DCM_PROTOCOL_DCMRXPDUID in combination DCM_PROTOCOL_RX_ADDR_TYP = physical can exist only once per “Module”)

The functional shall have a 1:n dependency to the DIAGNOSTIC_CONNECTION_TABLE.
(which means: DCM_PROTOCOL_DCMRXPDUID in combination DCM_PROTOCOL_RX_ADDR_TYP = functional can exist several times per “Module”)
The DIAGNOSTIC_PROTOCOL_TX_TABLE shall exist only once per “Module”

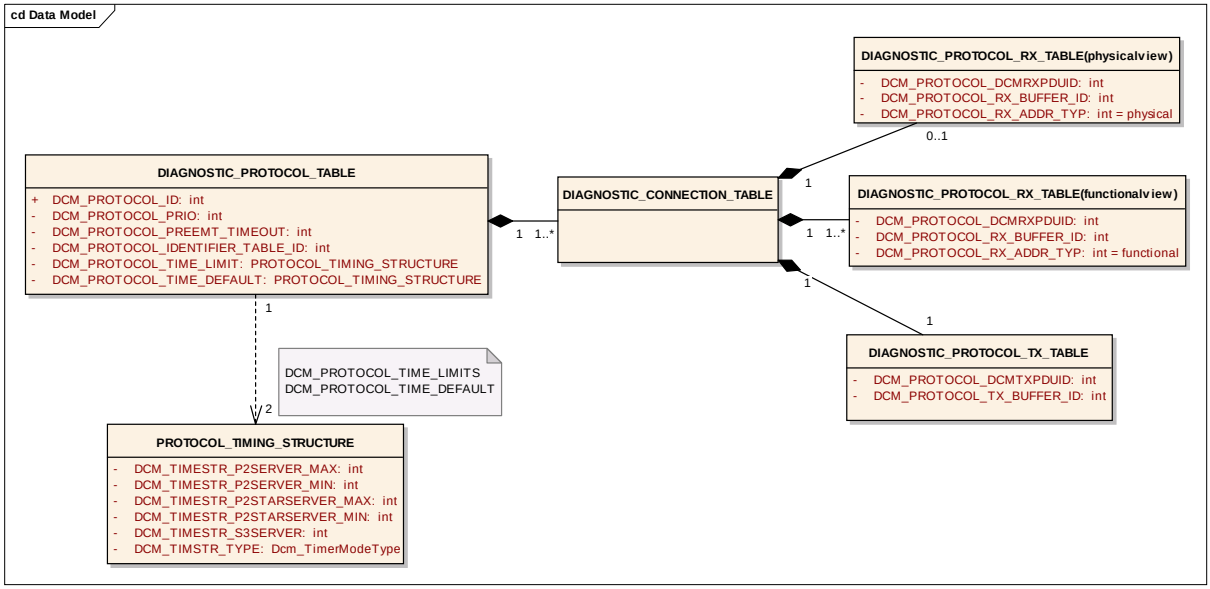


Figure 17 UML diagram of Protocol configuration

10.3.1.5 DIAGNOSTIC_BUFFER_CFG

SWS Item	Dcm032:
Container Name	DIAGNOSTIC_BUFFER_CFG
Description	This container contains the configuration (parameters) for the diagnostic buffer
Configuration Parameters	

Name	DCM_BUFFER_ID		
Description	Identifier of buffer		
Type	uint8		
Unit	--		
Range	0...max	max = maxsize uint8	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	none		

Name	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: 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		
Type	uint16		
Unit	bytes		
Range	8 – 4095	min. 8 byte max. 4095 byte (maximum number that can be represented on 12bits)	
	Pre-compile	x	Mandatory for all Variant

	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.3.1.6 PROTOCOL_TIMING_STRUCTURE

10.3.1.6.1 Row of PROTOCOL_TIMING_STRUCTURE:

SWS Item	Dcm031:		
Container Name	PROTOCOL_TIMING_STRUCTURE_ROW		
Description	Definition of a single Row of configuration for Protocol timing.		
Configuration Parameters			
Name	DCM_TIMSTR_TYPE		
Description	Type of the timing structure Note: The Dcm_TimerModeType DCM_CURRENT is not configurable		
Type	Dcm_TimerModeType		
Unit	--		
Range	DCM_DEFAULT		Configured Default timing parameters
	DCM_LIMITS		Configured Timing parameters limits
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	none		

Name	DCM_TIMSTR_P2SERVER_MIN		
Description	Value for $P2_{ServerMin}$ in ms		
Type	uint16		
Unit	ms		
Range	0…		
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	none		

Name	DCM_TIMSTR_P2SERVER_MAX		
Description	Value for $P2_{ServerMax}$ in ms		
Type	uint16		
Unit	ms		
Range	0…		
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		

Dependency	none

Name	DCM_TIMSTR_P2STARSERVER_MIN		
Description	Value for P2*$_{ServerMin}$ in ms		
Type	uint16		
Unit	ms		
Range	0…		
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	none		

Name	DCM_TIMSTR_P2STARSERVER_MAX		
Description	Value for P2*$_{ServerMax}$ in ms		
Type	uint16		
Unit	ms		
Range	0…		
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	none		

Name	DCM_TIMSTR_S3SERVER		
Description	Value for S3$_{Server}$ in ms		
Type	uint16		
Unit	ms		
Range	0…		
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	none		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.3.1.6.2 PROTOCOL_TIMING_STRUCTURE (consists of rows):

SWS Item	[Dcm031:]
Container Name	PROTOCOL_TIMING_STRUCTURE
Description	This container contains the configuration (parameters) for Protocol timing. Please notice that per protocol 2 Time tables shall be configured (DCM_DEFAULT and DCM_LIMITS) . This container contains Rows of PROTOCOL_TIMING_STRUCTURE_ROW
Configuration Parameters	

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
PROTOCOL_TIMING_STRUCTURE_ROW	2	ECU	--

10.3.1.7 RESPONSE_ON_EVENT_PARAMETERS

SWS Item	[Dcm069:]
Container Name	RESPONSE_ON_EVENT_PARAMETERS
Description	This container contains the configuration (parameters) for ResponseOnEvent service
Configuration Parameters	

Name	DCM_ROE_ENABLED		
Description	Allow to enable or disable a ROE transmission		
Type	#define		
Unit	--		
Range	CFG_ON		ROE handling enabled
	CFG_OFF		ROE handling disabled
Configuration Class	***Pre-compile***	x	Mandatory precompile parameter for all variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	DCM_ROE_TRANS_TYPE		
Description	Type of the used transmission (TYPE1 / TYPE2)		
Type	Enumeration		
Unit	--		
Range	TYPE1		transmission on the DcmTxPduId used for normal diagnostic responses (DCM_PROTOCOL_DCMTXPDUID)
	TYPE2		transmission on a separate DcmTxPduId
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_ROE_DCMTXPDUID		

Name	DCM_ROE_DCMTXPDUID		
Description	DcmTxPduId value for transmission of ROE responses (only needed for DCM_ROE_TRANS_TYPE TYPE2)		
Type	PduIdType		
Unit	--		
Range	0…max		max= number of DcmTxPduId-1
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_ROE_ENABLED		

	DCM_ROE_TRANS_TYPE

Name	DCM_ROE_TX_BUFFER_ID		
Description	Link to buffer configuration (DCM_BUFFER_ID) for configuration of ROE Tx buffer		
Type	uint8		
Unit	--		
Range	0…max	max = number of DCM_BUFFER_ID-1	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_ROE_ENABLED DCM_BUFFER_ID		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.3.1.8 PERIODIC_TRANSMISSION_PARAMETERS

SWS Item	[Dcm070:]
Container Name	PeriodicTransmissionParameters PERIODIC_TRANSMISSION_PARAMETERS
Description	This container contains the configuration (parameters) for Periodic Transmission service
Configuration Parameters	

Name	DCM_PERIODIC_TRANS_ENABLED		
Description	Allow to enable or disable a periodic transmission		
Type	#define		
Unit	--		
Range	CFG_ON		Periodic Transmission handling enabled
	CFG_OFF		Periodic Transmission handling disabled
Configuration Class	***Pre-compile***	x	Mandatory precompile parameter for all variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	DCM_PERIODIC_TRANS_TYPE		
Description	Type of the used transmission (TYPE1 / TYPE2)		
Type	Enumeration		
Unit	--		
Range	TYPE1	transmission on the DcmTxPduId used for normal diagnostic responses (DCM_PROTOCOL_DCMTXPDUID)	
	TYPE2	transmission on a separate DcmTxPduId	
	Pre-compile	x	Variant A

	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PERIODIC_TRANS_ENABLED		

Name	DCM_PERIODIC_TRANS_DCMTXPDUID		
Description	DcmTxPduId value for transmission of PeriodicTransmission responses (only needed for DCM_PERIODIC_TRANS_TYPE TYPE2)		
Type	PduIdType		
Unit	--		
Range	0…max	max= number of DcmTxPduId-1	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PERIODIC_TRANS_ENABLED DCM_PERIODIC_TRANS_TYPE		

Name	DCM_PERIODIC_TRANS_TX_BUFFER_ID		
Description	Link to buffer configuration (DCM_BUFFER_ID) for configuration of Periodic Transmission Tx buffer		
Type	uint8		
Unit	--		
Range	0…max	max = number of DCM_BUFFER_ID-1	
Configuration Class	***Pre-compile***	x	Variant A
	Link time	x	Variant B
	Post Build	L	Variant C
Scope	ECU		
Dependency	DCM_PERIODIC_TRANS_ENABLED DCM_BUFFER_ID		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.4 DSD (Diagnostic Service Dispatcher)

10.4.1 Configurable parameters

10.4.1.1 DCM_SIDTAB_SEC_LEVEL_ROW

SWS Item	--
Container Name	DCM_SIDTAB_SEC_LEVEL_ROW
Description	This container contains the configuration (DSD parameters) ”Security Level” included in the Service Identifier Table and in ReadDtc Sub Function Table
Configuration Parameters	

Name	DCM_SIDTAB_SEC_LEVEL
Description	Information about the Security Access Levels needed for execution of the DCM_SIDTAB_SERVICEID. Please refer to the ISO 14229-1, ISO 15031-5 and “Verification of the

	Security Access levels" (chapter 7.2.2.4). For DCM_SEC_LEVEL please refer to chapter 10.5 DSP (Diagnostic Service Processing) Please note, that it shall be provided to configure several DCM_SIDTAB_SEC_LEVEL per DCM_SIDTAB_SERVICEID		
Type	Dcm_SecLevelType		
Unit	- -		
Range	0x00, 0x01,0x03...0x7F	<configuration dependent – only odd values (and 0) are allowed (according to "securityAccessType" parameter of SecurityAccess request (see [6]))>	
	0xFF	all Sec. Access Levels	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	ECU		
Dependency	None		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.4.1.2 DCM_SIDTAB_SESSION_LEVEL_ROW

SWS Item	--
Container Name	DCM_SIDTAB_SESSION_LEVEL_ROW
Description	This container contains the configuration (DSD parameters) "Session Level" included in the Service Identifier Table
Configuration Parameters	

Name	DCM_SIDTAB_SESSION_LEVEL		
Description	Information about the Session Control needed for execution of the DCM_SIDTAB_SERVICEID. Please refer to the ISO 14229-1, ISO 15031-5 and "Verification of the Diagnostic Session" (chapter 7.2.2.4). For DCM_SESSION_LEVEL please refer to chapter 10.5 DSP (Diagnostic Service Processing) Please note, that it shall be provided to configure several DCM_SIDTAB_SESSION_LEVEL per DCM_SIDTAB_SERVICEID		
Type	Dcm_SesCtrlType		
Unit	- -		
Range	0..max	`max = See Dcm_SesCtrlType definition`	
	0xFF	all Session Control Levels	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	ECU		
Dependency	None		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.4.1.3 SERVICE_IDENTIFIER_TABLE

SWS Item	[Dcm071:]
Container Name	SERVICE_IDENTIFIER_TABLE
Description	This container contains the configuration (DSD parameters) for Service Identifier Table
Configuration Parameters	

Name	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		
Type	uint8		
Unit	- -		
Range	0…max	max = number of SERVICE_IDENTIFIER_TABLE -1	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	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		
Type	uint8		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	ECU		
Dependency	None		

Name	DCM_SIDTAB_FUNCTIONPOINTER		
Description	Function Pointer for the callback function of the DSP or/and the Autosar SW component for the particular DCM_SIDTAB_SERVICEID. 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*)		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	ECU		
Dependency	None		

Name	DCM_SIDTAB_SUBFUNC_AVAIL		
Description	Information whether the DCM_SIDTAB_SERVICEID 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.		
Type	Boolean		
Unit	--		
Range	TRUE	sub-function available	
	FALSE	sub-function not available	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	ECU		
Dependency	none		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DCM_SIDTAB_SESSION_LEVEL_ROW	0..n	ECU	--
DCM_SIDTAB_SEC_LEVEL_ROW	0..n	ECU	--

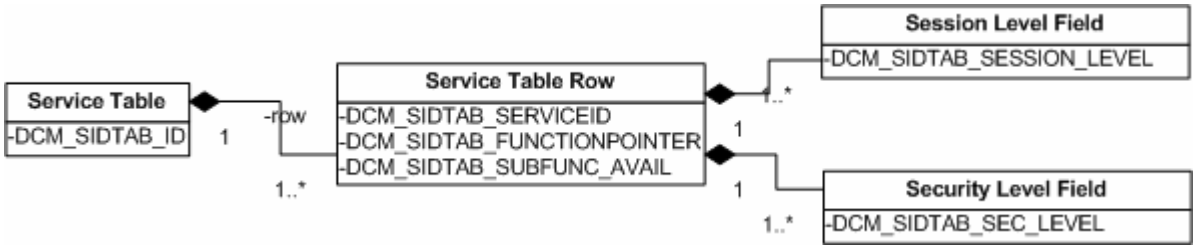


Figure 18 UML diagram of Service Identifier Table configuration

10.5 DSP (Diagnostic Service Processing)

10.5.1 Configurable parameters

10.5.1.1 DCM_SESSION_LEVEL_TABLE

10.5.1.1.1 Row of DCM_SESSION_LEVEL_TABLE:

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

Name	DCM_SESSION_LEVEL		
Description	hex value of the Session control.		
Type	Dcm_SesCtrlType		
Unit	`--`		
Range	`0..max`	max = See Dcm_SesCtrlType definition	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_SESSION_NAME		
Description	Name of the Session control		
Type	String		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_SESSION_P2STRSERVER_MAX		
Description	Value for P2*ServerMax in ms (per Session Control)		
Type or Unit	uint16		
Unit	`ms`		
Range	`0..`		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_SESSION_P2SERVER_MAX		
Description	Value for P2ServerMax in ms (per Session Control)		
Type or Unit	uint16		
Unit	`ms`		
Range	`0..`		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.5.1.1.2 DCM_SESSION_LEVEL_TABLE (consists of rows):

SWS Item	[Dcm072:]
Container Name	DCM_SESSION_LEVEL_TABLE
Description	This container contains the configuration (DSP parameter) session control configuration (per session control) This container contains Rows of DCM_SESSION_LEVEL_TABLE_ROW
Configuration Parameters	

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DCM_SESSION_LEVEL_TABLE_ROW	0..8	ECU	--

10.5.1.2 DCM_SEC_LEVEL_TABLE

10.5.1.2.1 Row of DCM_SEC_LEVEL_TABLE:

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

Name	DCM_SEC_LEVEL		
Description	hex value Security level value.		
Type	Dcm_SecLevelType		
Unit	`--`		
Range	`0x00, 0x01,0x03...0x7F`	<configuration dependent – only odd values (and 0) are allowed (according to “securityAccessType” parameter of SecurityAccess request (see [6]))>	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	X	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_SEC_NAME		
Description	Name of the security level		
Type	String		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	X	Variant A, Variant C
	Link time	X	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_SEC_NUM_MAX_ATT_DELAY
Description	Number of failed security accesses after which the delay time is

	activated		
Type or Unit	uint8		
Unit	--		
Range	0..		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	DCM_SEC_NUM_MAX_ATT_DELAY < DCM_SEC_NUM_MAX_ATT_LOCK		

Name	DCM_SEC_DELAY_INV_KEY		
Description	Delay time after failed security access (in ms). This is started after DCM_SEC_NUM_MAX_ATT_DELAY number of failed security accesses.		
Type or Unit	uint16		
Unit	Ms		
Range	0..		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_SEC_NUM_MAX_ATT_LOCK		
Description	Number of failed security accesses after which security access is locked		
Type or Unit	uint8		
Unit	--		
Range	0..		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	DCM_SEC_NUM_MAX_ATT_DELAY < DCM_SEC_NUM_MAX_ATT_LOCK		

Name	DCM_SEC_NUM_SEED		
Description	Size of seed in Bytes		
Type or Unit	uint8		
Unit	--		
Range	1..		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_SEC_NUM_KEY		
Description	Size of key in Bytes		
Type or Unit	uint8		
Unit	--		
Range	1..		
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
None	--	--	--

10.5.1.2.2 DCM_SEC_LEVEL_TABLE (consists of rows):

SWS Item	[Dcm074:]
Container Name	DCM_SEC_LEVEL_TABLE
Description	This container contains the configuration (DSP parameter) for security level configuration (per security level) This container contains Rows of DCM_SEC_LEVEL_TABLE_ROW
Configuration Parameters	

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DCM_SEC_LEVEL_TABLE_ROW	0..8	ECU	--

10.5.1.3 DCM_READDTC_SUB_FUNCTION_TABLE

10.5.1.3.1 Row of DCM_READDTC_SUB_FUNCTION_TABLE:

SWS Item	Dcm073:
Container Name	DCM_READDTC_SUB_FUNCTION_TABLE_ROW
Description	Definition of a single Row of configuration (parameters) of the service Read DTC Information – 0x19 (per sub-function type)
Configuration Parameters	

Name	DCM_DTC_INFO_SUB_FUNCTION_LEVEL		
Description	hex value of the sub-function.		
Type	uint8		
Unit	- -		
Range	0x01..0x	--	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	DCM_DTC_INFO_SUB_FUNCTION_SUPPORT
Description	Indicates whether the respective sub-function is supported or not.
Type	Boolean
Unit	--

Range	TRUE	sub-function is supported	
	FALSE	sub-function is not supported	
Configuration Class	***Pre-compile***	x	Variant A, Variant C
	Link time	x	Variant B
	Post Build	--	--
Scope	Module		
Dependency	None		

Included Containers			
Container Name	Multiplicity	Scope	Dependency
DCM_SIDTAB_SEC_LEVEL_ROW	0..n	ECU	--

10.5.1.3.2 DCM_READDTC_SUB_FUNCTION (consists of rows):

SWS Item	[Dcm074:]
Container Name	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 DCM_READDTC_SUB_FUNCTION_TABLE_ROW
Configuration Parameters	

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
DCM_READDTC_SUB_FUNCTION_TABLE_ROW	0..n (ISO 14229)	ECU	--

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.

SWS Item	Dcm013:	
Information elements		
Information element name	***Type / Range***	***Information element description***

DCM_VENDOR_ID	#define/ uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
DCM_MODULE_ID	#define/ uint8	Module ID of this module from Module List (DCM is 0x35)
DCM_AR_MAJOR_VERSION	#define/ uint8	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
DCM_AR_MINOR_VERSION	#define/ uint8	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
DCM_AR_PATCH_VERSION	#define/ uint8	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
DCM_SW_MAJOR_VERSION	#define/ uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
DCM_SW_MINOR_VERSION	#define/ uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
DCM_SW_PATCH_VERSION	#define/ uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

11 Changes to Release 1

11.1 Deleted SWS Items

SWS Item	*Rationale*
Not traced	Not traced

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
Not traced	Not traced	Not traced

11.3 Changed SWS Items

SWS Item	*Rationale*
Not traced	Not traced

11.4 Added SWS Items

SWS Item	*Rationale*
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