Document Title	Specification of Diagnostics
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	723

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-10

Document Change History

Date	Release	Changed by	Description
2018-10-31	18-10	AUTOSAR Release Management	• Diagnostic Protocol replaced by Diagnostic Conversations • ResponseOnEvent, CommunicationControl, EcuReset added • Chapter 7 overall rework and updates • Chapter 8 split into chapter 8 (C++ API) and chapter 9 (Service Interfaces)
2018-03-29	18-03	AUTOSAR Release Management	• Chapter 7.1. Software Cluster added • Chapter 7.2. Diagnostic Service Management, common parts for all services separated • Chapter 7.3. Event Management, several additions and rework • Chapter 8. API specification, complete rework
2017-10-27	17-10	AUTOSAR Release Management	• General API rework • TP Plug-in interface • Introduction of SoftwareCluster in APIs • Additional UDS services like SecurityAccess
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Adaptive Diagnostic Management (DM).

The DM is an UDS diagnostic implementation according to ISO 14229-1[1] for the Autosar Adaptive Platform. Unless stated otherwise in this document, the DM implements the functionality as defined in the ISO 14229-1[1]. Derivations, limitation, OEM or supplier-specific behaviour according to ISO 14229-1[1] are described in this document.

1.1 AUTOSAR Diagnostic Extract Template (DEXT)

The AUTOSAR Diagnostic Extract Template (DEXT) [2] is the configuration input to the DM.

1.2 Software Cluster

The AUTOSAR adaptive platform is able to be extended with new software packages without re-flashing the entire ECU. The individual software packages are described by *SoftwareCluster*s. To support the current approaches of diagnostic management (like software updates), each *SoftwareCluster* have its own DiagnosticAddresses.

DM is intended to support an own diagnostic server instance per installed *SoftwareCluster*. All diagnostic server instances share a single TransportLayer instance (e.g. DoIP on TCP/IP port 13400).

1.2.1 Diagnostic Server

The diagnostic service management response handling basically resembles the functionality of the Dcm BSW module of the AUTOSAR Classic platform. I.e. it is responsible for processing/dispatching of diagnostic services according to ISO 14229-1[1]. That means:

- Receiving UDS diagnostic request messages from the network layer
- Extracting transport layer independent UDS information from it.
- Dispatching the request towards the Diagnostic Server instances depending on target address and target address type (physical or functional) of received UDS request message
- Correlating the diagnostic request to an existing UDS session (if already exists)

- Checking whether the diagnostic request is allowed within current session and security settings
- If diagnostic request is NOT allowed, generate negative UDS response and send it to the network layer
- If diagnostic request is allowed, depending on DM's configuration and request type,
  - either process the service internally within diagnostic service handling function block of DM
  - or process the service internally within event memory management function block of DM
  - or hand it over for processing to an (external to DM) Adaptive Application

The figure below depicts those processing steps and functional blocks of DM's diagnostic service management part.

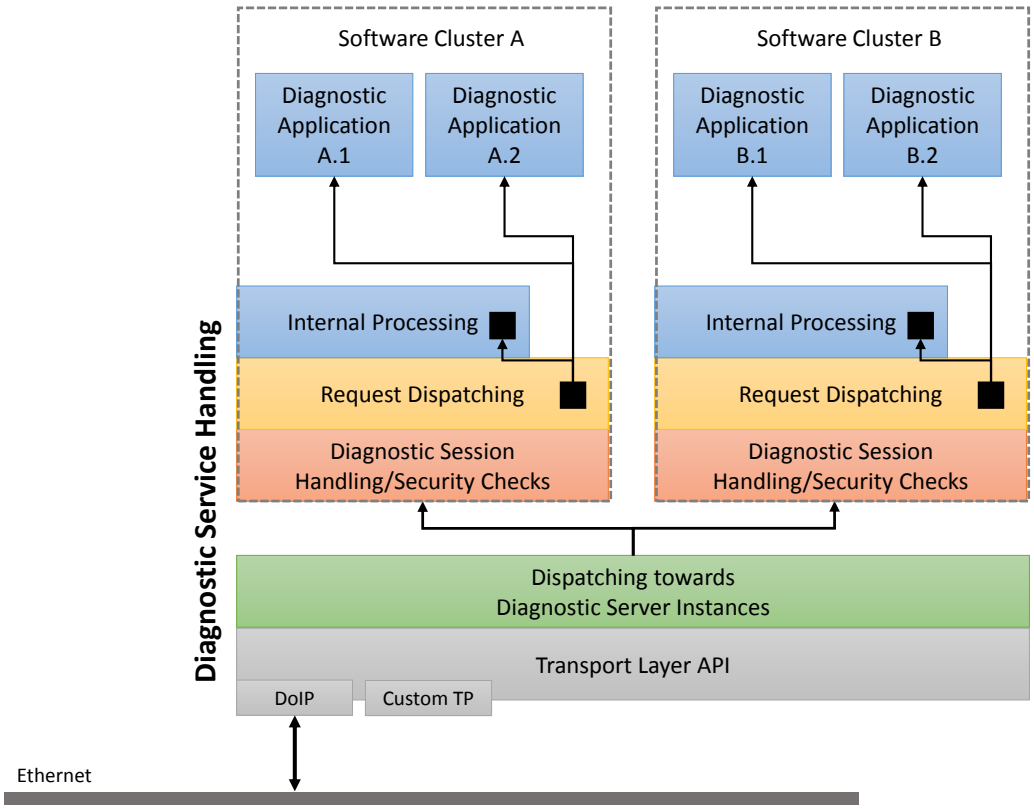

Figure 1.1: Architecture Diagnostic Service Handling

1.2.2 Diagnostic Managers external dependencies

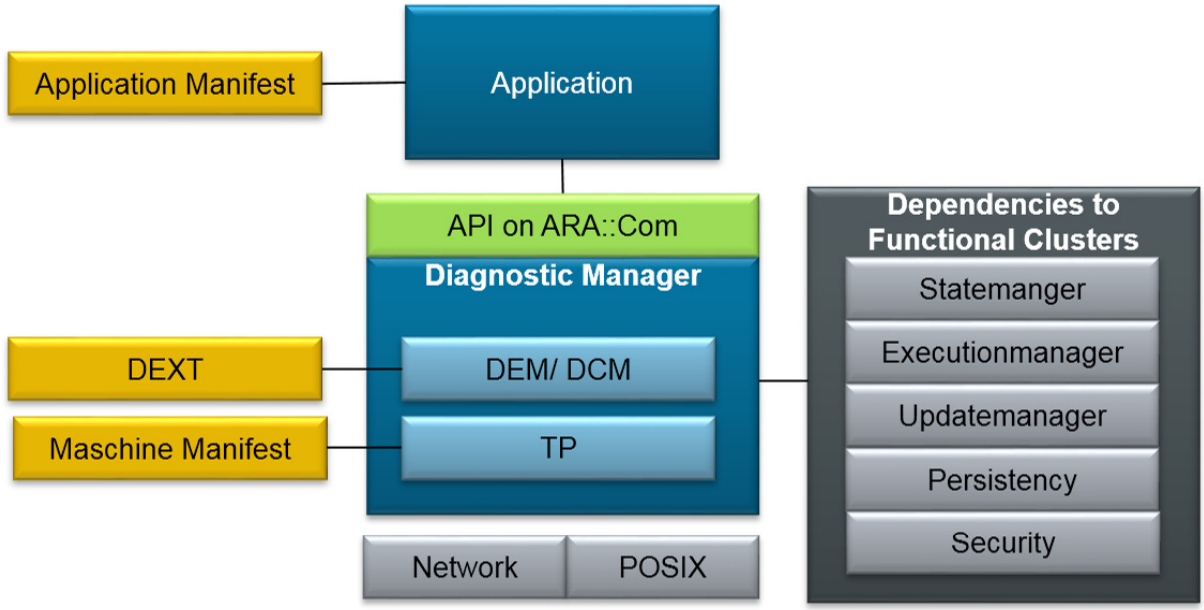

Figure 1.2: Diagnostic Managers external dependencies

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the DM module that are not included in the [3, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
AA	AUTOSAR Adaptive Application
AP	AUTOSAR Adaptive Platform
Channel	An abstraction of a network specific communication channel. In CAN networks a Channel can be identified via CAN identifier. In Ethernet networks a Channel might be defined by the quadruple Src-IP, Src-Port, Target-IP, Target-Port.
CP	AUTOSAR Classic Platform
DEXT	AUTOSAR Diagnostic Extract[2], describing diagnostic configuration of an ECU
DM	AUTOSAR Adaptive Diagnostic Management
DTC	Diagnostic Trouble Code according to ISO 14229-1[1]

Abbreviation / Acronym:	Description:
DID	Data Identified according to ISO 14229-1[1]. This 16 bit value uniquely defines one ore more data elements (parameters) that can are used in diagnostics to read, write or control data.
ECU	Electronic control unit
Execution Management	Functional cluster Execution Management
FDC	Fault Detection Counter according to 14229-1[1]
GID	Group identifier as used in DoIP
MetaInfo	Meta-Information in the form of a key-value map, which is given from DM to external service processors.
NRC	Negative Response Code used by UDS in the diagnostic response to indicate the tester that a certain failure has occurred and the diagnostic request was not processed.
PowerMode	Vehicle basic status information retrieval of DoIP
SA	Source Address of a UDS request
SID	Service Identifier, identifying a diagnostic service according to UDS, such as 0x14 ClearDiagnosticInformation
UDS	Unified Diagnostic Services
VIN	Vehicle Identification Number according to ISO-3779
Dcm	Diagnostic Communication Manager (Module of the AUTOSAR Classic Platform)
DoIP	Diagnostics over Internet Protocol (Communication protocol of automotive electronics according to ISO-13400[4])

Terms:	Description:
Aging	Unlearning/deleting of a no longer failed event/DTC after a defined number of operation cycles from event memory.
Associated ServiceInterface	Describes the association of a `ServiceInterface` to a `DiagnosticServiceSwMapping` by means of a referenced `SwcServiceDependency`, see section 7.2.4.2.1.
Diagnostic Conversation	Diagnostic Conversation represents a conversation between Diagnostic Client (Tester) and `Diagnostic Server`.
Diagnostic Management	Diagnostic Management is a placeholder for the complete functionality of diagnostic communication and event handling.
Diagnostic Server	DM is intended to support an own Diagnostic Server instance per installed *SoftwareCluster*, see section 7.2 for a detailed description.
DTC group	Uniquely identifies a set of `DTCs`. A DTC group is mapped to the range of valid DTCs. By providing a group of DTCs it is expressed that a certain operation is requested on all DTCs of that group. The DTC group definition is provided by ISO 14229-1[1] and OEM/supplier-specific.
Extended Data Records	Contains statistical data for a DTC. Extended data records are assigned to DTCs and maintained and stored by the DM.
Event	Uniquely identifies a fault path of the system. An application monitors the system and reports events to the DM.
Event memory	The DM stores information about events in the event memory. There can be multiple event memories, each keeping information independently from each other. Examples of the event memory is the UDS primary event memory or the up to 256 user-defined event memories.

Terms:	Description:
GroupOfAllDTCs	Identifies a special DTC group that contains all DTCs. This DTC group is identified by the DTC value 0xFFFFFF in 14229-1[1] and contains by default all DTCs of a fault memory. It is present by default in the DM and requires no configuration.
Internal, External	Classifies if a DiagnosticDataElement is either managed internally inside DM or by an external adaptive applications, see 7.2.4.1 for the precise definition.
Internally, Externally	Definition of the support type of a SID by the DM. Internally means processing is done by DM itself, Externally means an external service processor is used.
Monitor	A monitor is a piece of software running within an application, monitoring the correct functionality of a certain system part. The result of such a function check is reported to the DM in form of an diagnostic event.
Operation cycle	An operation cycle is the execution of monitor within an application, from a start point to a defined end point inside the application run.
Primary event memory	The primary event memory is used to store events and event related data. It is typically used by OEMs for after sales purposes, containing information to repair the vehicle.
Snapshot Record	Set of measurement values stored in the fault memory at a certain point of time during fault detection. It is used to gain environmental data information for occurred faults.
SoftwareCluster	A SoftwareCluster groups all AUTOSAR artifacts which are relevant to deploy software on a machine. This includes the definition of applications, i.e. their executables, application manifests, communication and diagnostics. In the context of diagnostics a SoftwareCluster can be addressed individually by its own set of diagnostic addresses.
UDS service	A diagnostic service as defined in ISO 14229-1[1].
UDS DTC status byte	Status byte as defined in ISO 14229-1[1], based on DTC level.
User-defined event memory	The user-defined event memory is used by the UDS service 0x19 with subfunctions 0x17, 0x18 and 0x19. It behaves as the primary event memory but contains data independent from the primary fault memory. It is used to store information that are relevant for different purposes such as warranty or development.

3 Related documentation

3.1 Input documents & related standards and norms

[1] Unified diagnostic services (UDS) – Part 1: Specification and requirements (Release 2013-03)
http://www.iso.org

[2] Diagnostic Extract Template
AUTOSAR_TPS_DiagnosticExtractTemplate

[3] Glossary
AUTOSAR_TR_Glossary

[4] Road vehicles – Diagnostic communication over Internet Protocol (DoIP)
http://www.iso.org

[5] General Specification of Adaptive Platform
AUTOSAR_SWS_General

[6] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement

[7] Specification of Communication Management
AUTOSAR_SWS_CommunicationManagement

[8] Specification of Log and Trace
AUTOSAR_SWS_LogAndTrace

[9] Specification of Persistency
AUTOSAR_SWS_Persistency

[10] Requirements on Diagnostics
AUTOSAR_SRS_Diagnostics

[11] Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part2: Network layer services

[12] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[13] Unified diagnostic services (UDS) - Part 2: Session layer services (Release 2013-03)
http://www.iso.org

[14] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

3.2 Further applicable specification

AUTOSAR provides a general specification [5] which is also applicable for Diagnostic Management. The specification SWS General shall be considered as additional and required specification for implementation of Diagnostic Management.

4 Constraints and assumptions

4.1 Known Limitations

This chapter describes known limitation of the DM in respect to general claimed goals of the module. The nature of constraints can be a general exclusion of a certain do-

main / functionality or it can be that the provided standard has not yet integrated this functionality and will do so in future releases.

- OBD ISO 15031 and WWH OBD ISO 27145 is not supported by the DM.
- *Software Cluster*/*Diagnostic Server instances* are supported by DM interfaces but are not specified in detail.
- *DoIP edge node* is not supported by the DM.
- The following UDS services are not implemented by the DM:
  - 0x23 ReadMemoryByAddress
  - 0x24 ReadScalingDataByIdentifier
  - 0x2A ReadDataByPeriodicIdentifier
  - 0x2C DynamicallyDefineDataIdentifier
  - 0x2F InputOutputControlByIdentifier
  - 0x38 RequestFileTransfer
  - 0x3D WriteMemoryByAddress
  - 0x83 AccessTimingParameter
  - 0x84 SecuredDataTransmission
  - 0x87 LinkControl
- Sub-functions of UDS services are implemented according to ISO 14229-1[1] unless explicitly stated.
- The UDS mirror event memory is not supported by the DM. As a result of this, the DM does not support the UDS service.
  - 0x19 with subfunction 0x0F (reportMirrorMemoryDTCByStatusMask)
  - 0x19 with subfunction 0x10 (reportMirrorMemoryDTCExtDataRecordBy-DTCNumber)
  - 0x19 with subfunction 0x11 (reportNumberOfMirrorMemoryDTCByStatus-Mask)
- The OBD/WWH OBD is not supported by the DM. As a result of this, the DM does not support the UDS service.
  - 0x19 with subfunction 0x05 (reportDTCStoredDataByRecordNumber)
  - 0x19 with subfunction 0x12 (reportNumberOfEmissionsOBDDTCByStatus-Mask)
  - 0x19 with subfunction 0x13 (reportEmissionsOBDDTCByStatusMask)
  - 0x19 with subfunction 0x42 (reportWWHOBDDTCByMaskRecord)

  - 0x19 with subfunction 0x55 (reportWWHOBDDTCWithPermanentStatus)
- Event Memory: Variant handling at runtime for events/DTCs is not supported.
- Event Memory: Details for combined events are not specified.
- Event Memory: Event displacement is not supported. The DM stores for each DTC related data.
- Event Memory: Interface to read the number of event memory entries is not supported.
- Event Memory: Internal configuration parameters and DM values as extended data are not supported.

5 Dependencies to other modules

As any other process started by Execution Management [6], DM needs to interact with the Execution Management.

DM is a service and therefore uses ara::com [7] to communicate with applications.

The DM may use ara::log ([8], Log and Trace) for logging and tracing purposes.

DM may use ara::per ([9], Persistency) to store non-volatile data.

6 Requirements Tracing

The following tables reference the requirements specified in [10] and links to the fulfilling requirements by this document. Please note that the column “Satisfied by” being empty for a specific requirement means that the requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_Diag_04005]	Manage Security Access level handling	[SWS_DM_00047] [SWS_DM_00103] [SWS_DM_00236] [SWS_DM_00421] [SWS_DM_00427] [SWS_DM_00428] [SWS_DM_00429] [SWS_DM_00430]
[SRS_Diag_04006]	Manage session handling	[SWS_DM_00046] [SWS_DM_00101] [SWS_DM_00102] [SWS_DM_00380] [SWS_DM_00381] [SWS_DM_00382] [SWS_DM_00383] [SWS_DM_00435]
[SRS_Diag_04016]	Support ”Busy handling” by sending a negative response 0x78	[SWS_DM_00368] [SWS_DM_00369]
[SRS_Diag_04019]	Provide confirmation after transmit diagnostic responses to the application	[SWS_DM_00268] [SWS_DM_00341]

Requirement	Description	Satisfied by
[SRS_Diag_04020]	Suppress responses to diagnostic tool requests	[SWS_DM_00365] [SWS_DM_00366] [SWS_DM_00433]
[SRS_Diag_04033]	Support the upload/download services for reading/writing data in an ECU in an extended and manufacturer specific diagnostic session	[SWS_DM_00128] [SWS_DM_00136] [SWS_DM_00138] [SWS_DM_00139] [SWS_DM_00142] [SWS_DM_00143]
[SRS_Diag_04059]	Configuration of timing parameters	[SWS_DM_NA]
[SRS_Diag_04064]	Provide configurable buffer sizes for storage of the events, status information and environmental data	[SWS_DM_NA]
[SRS_Diag_04067]	Provide the diagnostic status information according to ISO 14229-1	[SWS_DM_00061] [SWS_DM_00062] [SWS_DM_00063] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374]
[SRS_Diag_04068]	The diagnostic in AUTOSAR shall support event specific debounce counters to improve signal quality internally (According to ISO 14229-1 Appendix D)	[SWS_DM_00013] [SWS_DM_00014] [SWS_DM_00015] [SWS_DM_00026] [SWS_DM_00030] [SWS_DM_00031] [SWS_DM_00032] [SWS_DM_00033] [SWS_DM_00034] [SWS_DM_00035] [SWS_DM_00036] [SWS_DM_00037] [SWS_DM_00038] [SWS_DM_00039] [SWS_DM_00040] [SWS_DM_00085] [SWS_DM_00086] [SWS_DM_00089]
[SRS_Diag_04097]	Decentralized and modular diagnostic configuration in applications	[SWS_DM_00393] [SWS_DM_00397] [SWS_DM_00401] [SWS_DM_00404] [SWS_DM_00407] [SWS_DM_00408] [SWS_DM_00418] [SWS_DM_00503] [SWS_DM_00504] [SWS_DM_00505] [SWS_DM_00506] [SWS_DM_00508] [SWS_DM_00509] [SWS_DM_CONSTR_00394] [SWS_DM_CONSTR_00395] [SWS_DM_CONSTR_00396]
[SRS_Diag_04109]	Provide an interface to retrieve the number of event memory entries	[SWS_DM_00651]
[SRS_Diag_04115]	The optional parameter DTCSettingControlOption Record as part of UDS service ControlDTCSetting shall be limited to GroupOfDTC	[SWS_DM_00231]

Requirement	Description	Satisfied by
[SRS_Diag_04117]	Configurable behavior for DTC deletion	[SWS_DM_00065] [SWS_DM_00091] [SWS_DM_00092] [SWS_DM_00116] [SWS_DM_00117] [SWS_DM_00121] [SWS_DM_00122] [SWS_DM_00123] [SWS_DM_00124] [SWS_DM_00144] [SWS_DM_00145] [SWS_DM_00146] [SWS_DM_00147] [SWS_DM_00159] [SWS_DM_00160] [SWS_DM_00481] [SWS_DM_CONSTR_00082]
[SRS_Diag_04119]	Handle the execution of diagnostic services according to the assigned diagnostic session	[SWS_DM_00046]
[SRS_Diag_04120]	Support a predefined Address AndLengthFormatIdentifier	[SWS_DM_00129] [SWS_DM_00130]
[SRS_Diag_04124]	Store the current debounce counter value non-volatile to over a power-down cycle	[SWS_DM_00018] [SWS_DM_00028]
[SRS_Diag_04125]	Event debounce counter shall be configurable	[SWS_DM_00017] [SWS_DM_00019] [SWS_DM_00020] [SWS_DM_00021] [SWS_DM_00022] [SWS_DM_00023] [SWS_DM_00024] [SWS_DM_00025] [SWS_DM_00029]
[SRS_Diag_04127]	Configurable record numbers and trigger options for DTCSnapshotRecords and DTCExtendedDataRecords	[SWS_DM_00153] [SWS_DM_00156]
[SRS_Diag_04133]	Aging for event memory entries	[SWS_DM_00237] [SWS_DM_00238] [SWS_DM_00239] [SWS_DM_00240] [SWS_DM_00241] [SWS_DM_00242]
[SRS_Diag_04140]	Aging for UDS status bits ”confirmedDTC” and ”testFailed SinceLastClear”	[SWS_DM_00243]
[SRS_Diag_04148]	Provide capabilities to inform applications about diagnostic data changes	[SWS_DM_00273]
[SRS_Diag_04150]	Support the primary fault memory defined by ISO 14229-1	[SWS_DM_00056] [SWS_DM_00083] [SWS_DM_CONSTR_00084]
[SRS_Diag_04151]	Event status handling	[SWS_DM_00213] [SWS_DM_00214] [SWS_DM_00215]
[SRS_Diag_04157]	Reporting of DTCs and related data	[SWS_DM_00061] [SWS_DM_00062] [SWS_DM_00063] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00247] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374]
[SRS_Diag_04159]	Control of DTC storage	[SWS_DM_00088] [SWS_DM_00229] [SWS_DM_00232] [SWS_DM_00233] [SWS_DM_00378]

Requirement	Description	Satisfied by
[SRS_Diag_04160]	ResponseOnEvent according to ISO 14229-1	[SWS_DM_00491] [SWS_DM_00492] [SWS_DM_00493] [SWS_DM_00494] [SWS_DM_00495] [SWS_DM_00496] [SWS_DM_00497] [SWS_DM_00498] [SWS_DM_00499] [SWS_DM_00500] [SWS_DM_00501]
[SRS_Diag_04166]	Several tester conversations in parallel with assigned priorities	[SWS_DM_00011] [SWS_DM_00016] [SWS_DM_00422] [SWS_DM_00423] [SWS_DM_00424] [SWS_DM_00425] [SWS_DM_00426] [SWS_DM_00432] [SWS_DM_00484]
[SRS_Diag_04167]	Conversation preemption/abortion	[SWS_DM_00042] [SWS_DM_00049] [SWS_DM_00277] [SWS_DM_00278] [SWS_DM_00279] [SWS_DM_00280] [SWS_DM_00290] [SWS_DM_00431] [SWS_DM_00482] [SWS_DM_00483] [SWS_DM_00485]
[SRS_Diag_04168]	Adding of user-defined transport layers	[SWS_DM_00329] [SWS_DM_00330] [SWS_DM_00331] [SWS_DM_00332] [SWS_DM_00333] [SWS_DM_00340] [SWS_DM_00342] [SWS_DM_00345] [SWS_DM_00346] [SWS_DM_00347] [SWS_DM_00348] [SWS_DM_00349] [SWS_DM_00350] [SWS_DM_00351] [SWS_DM_00356] [SWS_DM_00357] [SWS_DM_00358] [SWS_DM_00359] [SWS_DM_00385] [SWS_DM_00386] [SWS_DM_00387] [SWS_DM_00388] [SWS_DM_00389] [SWS_DM_00392] [SWS_DM_00487]
[SRS_Diag_04169]	Provide an interface for external UDS service processors.	[SWS_DM_00197]
[SRS_Diag_04171]	Synchronous and asynchronous interaction with external service processors	[SWS_DM_NA]
[SRS_Diag_04172]	Inform external service processors about outcome of the final response	[SWS_DM_00341]
[SRS_Diag_04178]	Support operation cycles according to ISO 14229-1	[SWS_DM_00002] [SWS_DM_00003] [SWS_DM_00004] [SWS_DM_00167] [SWS_DM_00169] [SWS_DM_00192] [SWS_DM_00216] [SWS_DM_CONSTR_00168]
[SRS_Diag_04179]	Provide interfaces for monitoring application.	[SWS_DM_00007] [SWS_DM_00008] [SWS_DM_00168]

Requirement	Description	Satisfied by
[SRS_Diag_04180]	Process all UDS Services related to diagnostic fault memory of ISO 14229-1 internally	[SWS_DM_00062] [SWS_DM_00090] [SWS_DM_00091] [SWS_DM_00092] [SWS_DM_00104] [SWS_DM_00115] [SWS_DM_00162] [SWS_DM_00163] [SWS_DM_00164] [SWS_DM_00165] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00229] [SWS_DM_00232] [SWS_DM_00233] [SWS_DM_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00247] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374]
[SRS_Diag_04183]	Notify interested parties about event status changes	[SWS_DM_00219] [SWS_DM_00220]
[SRS_Diag_04185]	Notify applications about the clearing of an event	[SWS_DM_00067]
[SRS_Diag_04186]	Notify applications about the start or restart of an operation cycle	[SWS_DM_00068] [SWS_DM_00069] [SWS_DM_00070] [SWS_DM_00071]
[SRS_Diag_04189]	Support a fine grained configuration for Snapshot Records and ExtendedData Records	[SWS_DM_00151] [SWS_DM_00155]
[SRS_Diag_04190]	Usage of internal data elements in SnapshotRecords and ExtendedDataRecords	[SWS_DM_00152] [SWS_DM_00154]
[SRS_Diag_04192]	Provide the ability to handle event specific enable conditions	[SWS_DM_00074] [SWS_DM_00087] [SWS_DM_00377] [SWS_DM_00379]
[SRS_Diag_04194]	ClearDTC shall be accessible for applications	[SWS_DM_00260] [SWS_DM_00261] [SWS_DM_00262] [SWS_DM_00263] [SWS_DM_00265] [SWS_DM_00266] [SWS_DM_00267]
[SRS_Diag_04195]	Chronological reporting order of the DTCs located in the configured event memory	[SWS_DM_NA]
[SRS_Diag_04196]	UDS Service handling for all diagnostic services defined in ISO 14229-2	[SWS_DM_00090] [SWS_DM_00096] [SWS_DM_00097] [SWS_DM_00104] [SWS_DM_00113] [SWS_DM_00114] [SWS_DM_00126] [SWS_DM_00127] [SWS_DM_00128] [SWS_DM_00131] [SWS_DM_00134] [SWS_DM_00137] [SWS_DM_00140] [SWS_DM_00141] [SWS_DM_00162] [SWS_DM_00170] [SWS_DM_00177] [SWS_DM_00186] [SWS_DM_00198] [SWS_DM_00199] [SWS_DM_00201] [SWS_DM_00210] [SWS_DM_00211] [SWS_DM_00212] [SWS_DM_00227] [SWS_DM_00234] [SWS_DM_00235] [SWS_DM_00236] [SWS_DM_00269] [SWS_DM_00360] [SWS_DM_00361] [SWS_DM_00363] [SWS_DM_00364] [SWS_DM_00367] [SWS_DM_00376] [SWS_DM_00419]

Requirement	Description	Satisfied by
[SRS_Diag_04197]	Clearing the user defined fault memory	[SWS_DM_00193] [SWS_DM_00194] [SWS_DM_00195] [SWS_DM_00208]
[SRS_Diag_04198]	Process all UDS Services related to session and security management of ISO 14229 internally	[SWS_DM_00104] [SWS_DM_00226] [SWS_DM_00228]
[SRS_Diag_04199]	Provide a configurable UDS service execution mechanism at runtime to decide if a UDS request shall be processed or not	[SWS_DM_00106] [SWS_DM_00108] [SWS_DM_00111] [SWS_DM_00112] [SWS_DM_00286] [SWS_DM_00287] [SWS_DM_00288] [SWS_DM_00289]
[SRS_Diag_04200]	Support event combination	[SWS_DM_NA]
[SRS_Diag_04201]	Support a configuration to assign specific events to a customer specific DTC	[SWS_DM_00060] [SWS_DM_CONSTR_00059]
[SRS_Diag_04202]	Report DTCs getting active to the error logging module/system	[SWS_DM_NA]
[SRS_Diag_04203]	Common checks on all supported UDS Services Requests	[SWS_DM_00096] [SWS_DM_00098] [SWS_DM_00099] [SWS_DM_00100] [SWS_DM_00101] [SWS_DM_00102] [SWS_DM_00103] [SWS_DM_00202] [SWS_DM_00203] [SWS_DM_00230] [SWS_DM_00231] [SWS_DM_00249] [SWS_DM_00252] [SWS_DM_00362] [SWS_DM_00409] [SWS_DM_00412] [SWS_DM_00413] [SWS_DM_00414] [SWS_DM_00415] [SWS_DM_00416] [SWS_DM_00417] [SWS_DM_00437] [SWS_DM_00438] [SWS_DM_00439] [SWS_DM_00440] [SWS_DM_00441] [SWS_DM_00442] [SWS_DM_00443] [SWS_DM_00444] [SWS_DM_00445] [SWS_DM_00446] [SWS_DM_00447] [SWS_DM_00448] [SWS_DM_00507]
[SRS_Diag_04204]	Provide the current status of each warning indicator.	[SWS_DM_00221] [SWS_DM_00222] [SWS_DM_00223] [SWS_DM_00224]
[SRS_Diag_04205]	Support of SnapshotRecords	[SWS_DM_00151] [SWS_DM_00152] [SWS_DM_00153]
[SRS_Diag_04206]	Support of ExtendedData Records	[SWS_DM_00154] [SWS_DM_00155] [SWS_DM_00156]
[SRS_Diag_04208]	Inform the application about diagnostic session and diagnostic security level changes on each tester connection.	[SWS_DM_00248] [SWS_DM_00250] [SWS_DM_00270] [SWS_DM_00271] [SWS_DM_00272] [SWS_DM_00478] [SWS_DM_00479] [SWS_DM_00480] [SWS_DM_CONSTR_00208]
[SRS_Diag_04209]	Pseudo parallel client interaction according to ISO	[SWS_DM_00011]
[SRS_Diag_04210]	Fully parallel client interaction	[SWS_DM_00011]
[SRS_Diag_04211]	Persistent storage of DTC status and environmental data	[SWS_DM_00148] [SWS_DM_00150]
[SRS_Diag_04214]	Support the user defined fault memories defined by ISO 14229-1	[SWS_DM_00055] [SWS_DM_00057]

Requirement	Description	Satisfied by
[SRS_Diag_04216]	Support for multiple Diagnostic Server Instances	[SWS_DM_00390] [SWS_DM_00391] [SWS_DM_00420]
[SRS_Diag_04218]	Support of UDS service 0x2F InputOutputControlByIDentifier	[SWS_DM_NA]
[SRS_Eth_00026]	No description	[SWS_DM_00205] [SWS_DM_00434] [SWS_DM_00449] [SWS_DM_CONSTR_00206]
[SRS_Eth_00027]	No description	[SWS_DM_00449]
[SRS_Eth_00080]	No description	[SWS_DM_00449]
[SRS_Eth_00082]	No description	[SWS_DM_00449]
[SRS_Eth_00083]	No description	[SWS_DM_00005] [SWS_DM_00449]
[SRS_Eth_00084]	No description	[SWS_DM_00449]
[TPS_DEXT_01008]	No description	[SWS_DM_00058]
[TPS_DEXT_03014]	No description	[SWS_DM_00064]

6.1 Not applicable requirements

[SWS_DM_NA] ⌈ These requirements are not applicable as they are not within the scope of this release. ⌋*(SRS_Diag_04059, SRS_Diag_04064, SRS_Diag_04171, SRS_Diag_04195, SRS_Diag_04200, SRS_Diag_04202, SRS_Diag_04218)*

7 Functional specification

The functionality of DM is split into two layers: the UDS Transport Layer and the Application Layer. On the UDS Transport Layer, DM handles connections to Diagnostic Clients via standardized or user defined UDS Transport Protocols, see section 7.1 for details. The subcomponent of DM implementing a particular Transport Protocol is called a Transport Protocol Handler.

On the Application Layer, DM implements the two main building blocks of diagnostics: event memory management and diagnostic service handling, both according to UDS ISO 14229-1[1]. On AUTOSAR adaptive platform the Application Layer can be split into multiple SoftwareClusters, each with its own diagnostic address. Accordingly, DM instantiates for each SoftwareCluster a Diagnostic Server that implements diagnostics with scope given by this SoftwareCluster, see section 7.2.

The link between the UDS Transport Layer and the the Application Layer is implemented by the Transport Protocol Manager, which dispatches UDS messages in both directions: UDS requests from Diagnostic Clients are forwarded to the respective responsible Diagnostic Server Instance, and UDS responses created by Diagnostic Server Instance are dispatched towards the respective Transport Protocol Handler that handles the connection to the Diagnostic Client.

A broad subcomponent view on DM is given as follows:

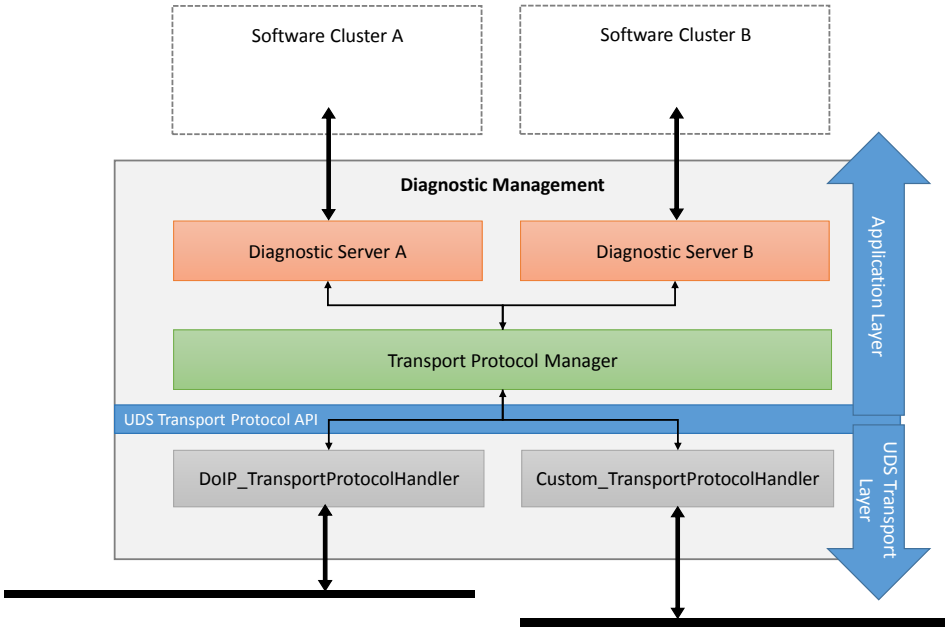

Figure 7.1: Component view on Diagnostic Management

7.1 UDS Transport Layer

Since there exist standardized as well as OEM specific UDS Transport Layers, the DM supports a standardized C++ API (called `Transport Protocol API`) where different kinds of UDS Transport Layers can be connected. Currently the Adaptive Platform only provides a detailed description of Ethernet-based network technologies, which mandates support of DoIP [4]. It is very likely, that upcoming releases of the DM will also detail CAN, CAN-FD, FR, ... networks. The `Transport Protocol API` allows for extensions of DM towards not-yet-detailed and proprietary UDS Transport Protocols.

7.1.1 Support of proprietary UDS Transport Layer

The UDS Transport Protocol API is formally described in section 8.1. This section describes the required interaction of the components using this API. Each (proprietary) `UDS Transport Protocol` implementation subclasses the abstract class `UdsTransportProtocolHandler`, which shall be provided by DM according to [SWS_DM_00315].

7.1.1.1 Initialization, Starting and Stopping of a proprietary UDS TransportLayer

[SWS_DM_00329] Lifecycle management of an `Uds Transport Protocol` implementation ⌈ The lifecycle of an `Uds Transport Protocol` implementation shall be managed by the `DM` in the following order:

- Creation of `Uds Transport Protocol` implementation by calling its constructor (see [SWS_DM_09015]).
- Initializing of `Uds Transport Protocol` implementation by calling `Initialize` (see [SWS_DM_00319])
- Starting of `Uds Transport Protocol` implementation by calling `Start` (see [SWS_DM_00322])
- Stopping of `Uds Transport Protocol` implementation by calling `Stop` (see [SWS_DM_00323])

⌋*(SRS_Diag_04168)*

[SWS_DM_00330] Construction of an `Uds Transport Protocol` implementation ⌈ The `DM` shall call the specific constructor of the `Uds Transport Protocol` implementation, where the argument `handler_id` is unique among all by `DM` instantiated `Uds Transport Protocol` implementations and the `transport_protocol_mgr` is set to the reference of the instance of `UdsTransportProtocolMgr` (see [SWS_DM_00306]) provided by `DM`. ⌋*(SRS_Diag_04168)*

[SWS_DM_00331] Initialization of an `Uds Transport Protocol` implementation ⌈ The `DM` shall call the `Initialize` (see [SWS_DM_00319]) method of the `Uds Transport Protocol` implementation during startup/initialization phase, before reporting `ApplicationState.kRunning` to the execution management. ⌋*(SRS_Diag_04168)*

[SWS_DM_00332] Starting of an `Uds Transport Protocol` implementation ⌈ The `DM` shall call the `Start` (see [SWS_DM_00322]) method of the `Uds Transport Protocol` implementation during startup/initialization phase, before reporting `ApplicationState.kRunning` to the execution management and after call to `Initialize` has returned. ⌋*(SRS_Diag_04168)*

[SWS_DM_00333] Stopping of an `Uds Transport Protocol` implementation ⌈ The `DM` shall call the `Stop` (see [SWS_DM_00323]) method of each `Uds Transport Protocol` implementation, it has started, if it is switching to state `ApplicationState.kTerminating`. ⌋*(SRS_Diag_04168)*

[SWS_DM_00340] Waiting for Stop confirmation ⌈ After having called `Stop` method of any `Uds Transport Protocol` implementation, it shall wait for the corresponding `HandlerStopped` (see [SWS_DM_00314]) callback with the related `handler_id`, before it finally terminates the process. ⌋*(SRS_Diag_04168)*

7.1.1.2 UDS message reception on a proprietary UDS TransportLayer

[SWS_DM_00342] Indication of UDS message reception ⌈ `Uds Transport Protocol` implementation shall call `IndicateMessage` ([SWS_DM_00309]) on its `UdsTransportProtocolMgr` reference ((see [SWS_DM_00330])), as soon as it has at least the following information of an incoming UDS request available:

- UDS source address of the request.
- UDS target address of the request.
- Type of the UDS target address (physical or functional)
- Size of the entire UDS message starting from SID

⌋*(SRS_Diag_04168)*

[SWS_DM_00347] Channel identification in Indication ⌈ `Uds Transport Protocol` implementation shall determine a distinct identifier to identify the network specific channel over which the UDS request has been received, which can be later used to deliver the UDS response to the source of the UDS request. ⌋*(SRS_Diag_04168)*

Note: A diagnostic client has basically two address parts which together serve for its unique identification:

- The UDS source address (SA) in the clients/testers request which represent a technology/transport layer independent part.
- The technology/transport layer specific/dependent network endpoint source address, from which the request from the client originates. In Ethernet-based networks this typically is an IP-address/port number pair, while in CAN networks it is the CAN identifier of the CAN-TP message used by the client. In UDS on CAN (ISO ISO-15765-2[11]) contrary to DoIP, the SA is not explicitly transmitted, but directly deduced from the CAN identifier of the CAN-TP message. That means on CAN we do not have two separate address parts, only the network endpoint source address part is used for identification.

The side effect of this is that from the viewpoint of `Diagnostic Server`, which supports parallel Diagnostic Clients, it is a perfectly valid scenario that two Diagnostic Clients with the same UDS `SA` can be active in parallel if they originate from different/distinguishable network endpoints.

[SWS_DM_00385] Acceptance of UDS message reception ⌈ If the `DM` is able to process the indicated request, it shall return a `std::pair` with `IndicationResult` set to `kIndicationOk` and a `UdsMessagePtr`, which owns a valid `UdsMessage` object, with a capacity of so many bytes, the `DM` wants to process of the indicated request. The minimum size of the `UdsMessage` object shall be one byte. ⌋*(SRS_Diag_04168)*

[SWS_DM_00392] Properties of returned UdsMessage ⌈ If the `DM` accepted the UDS message reception, the returned `UdsMessage` owned by `UdsMessagePtr` shall return a ByteVector from GetPayload, which shall be empty (i.e. `empty()` returns true, `size()` returns 0). ⌋*(SRS_Diag_04168)*

Note: In the normal case, where DM accepts the complete UDS request for processing, it will provide a std::pair with IndicationResult set to kIndicationOk and a UdsMessagePtr, which owns a valid UdsMessage object, with the capacity equal (or greater) to parameter Size indicated by Uds Transport Protocol implementation. There are use cases (typically for negative responses), where the DM does NOT need the entire UDS request message data to generate the UDS response and therefore might return a UdsMessagePtr, which owns a valid UdsMessage object, with a capacity smaller than the indicated parameter Size. E.g. this is useful e.g. in the case, where DM is busy and wants to ignore/reject a second parallel request. For declining a second request WITH sending a negative response according to [SWS_DM_00049], the DM would return an UdsMessagePtr with only enough capacity to be able to construct a valid negative response.

[SWS_DM_00386] Ignoring UDS message reception because DM is busy ⌈ If the DM is busy and not able to process the indicated UDS request, it shall return a std::pair with IndicationResult set to kIndicationOccupied and a UdsMessagePtr equal to UdsMessagePtr(nullptr). ⌋*(SRS_Diag_04168)*

Note: For declining/ignoring a second request without sending a negative response according to [SWS_DM_00290], the DM would choose this behavior.

[SWS_DM_00387] Ignoring UDS message reception because DM has no (memory) ressources ⌈ If the DM is not able to process the indicated UDS request, because it has not enough (memory) resources to hold the indicated UDS request, it shall return a std::pair with IndicationResult set to kIndicationOverflow and a UdsMessagePtr equal to UdsMessagePtr(nullptr). ⌋*(SRS_Diag_04168)*

Note: There might exist Uds Transport Protocol implementations, which make NO distinction between [SWS_DM_00386] and [SWS_DM_00387]. I.e. regardless, whether the DM returns a kIndicationOverflow or kIndicationOccupied, the behavior on transport layer level is the same. But, for instance, a CanTP Uds Transport Protocol implementation, would explicitly react on a kIndicationOverflow with sending a FC.OFLW on CanTP level to the UDS request sender.

[SWS_DM_00487] Ignoring UDS message reception because of unknown target address ⌈ If the DM is not able to process the indicated UDS request, because the indicated target address is unknown to DM, it shall return a std::pair with IndicationResult set to kIndicationUnknownTargetAddress and a UdsMessagePtr equal to UdsMessagePtr(nullptr). ⌋*(SRS_Diag_04168)*

[SWS_DM_00388] Filling provided UdsMessage ⌈ If the DM returned kIndicationOK from the IndicateMessage, the Uds Transport Protocol implementation shall fill the UdsMessage owned by UdsMessagePtr from the received UDS request starting from SID up to either UdsMessage full capacity or up to the entire received UDS request message, whatever happens first. ⌋*(SRS_Diag_04168)*

[SWS_DM_00345] Forwarding of UDS message ⌈ If the Uds Transport Protocol implementation has filled the payload of the returned UdsMessagePtr, it shall call HandleMessage ([SWS_DM_00311]) on its UdsTransportProtocolMgr ref-

erence ((see [SWS_DM_00330]) with the returned UdsMessagePtr as argument. ⌋ *(SRS_Diag_04168)*

[SWS_DM_00389] Skipping Forwarding of UDS message ⌈ If the DM returned a IndicationResult NOT equal to kIndicationOK from the IndicateMessage, the Uds Transport Protocol implementation shall NOT call HandleMessage. ⌋ *(SRS_Diag_04168)*

[SWS_DM_00346] Aborting of UDS message ⌈ If the Uds Transport Protocol implementation has already called IndicateMessage (see [SWS_DM_00342]), but is not willing to call HandleMessage (maybe due to errors receiving the entire/remaining UDS request), it shall notify DM by calling NotifyMessageFailure ([SWS_DM_00310]) on its UdsTransportProtocolMgr reference ((see [SWS_DM_00330]) with the returned UdsMessagePtr as argument. ⌋ *(SRS_Diag_04168)*

7.1.1.3 UDS message transmission on a proprietary UDS TransportLayer

[SWS_DM_00348] Transmission of UDS response message ⌈ DM shall send a diagnostic response UDS message to the same Uds Transport Protocol implementation, where it has received the UDS request message (see [SWS_DM_00345]) by calling the Transmit (see [SWS_DM_00327]) method of the Uds Transport Protocol implementation. ⌋*(SRS_Diag_04168)*

[SWS_DM_00349] Reuse channel identifier of Indication ⌈ DM shall set the argument channel_id in the Transmit call to the same value as in the Indication of the corresponding UDS request message (see [SWS_DM_00347]). ⌋*(SRS_Diag_04168)*

[SWS_DM_00350] Confirmation of UDS message transmission ⌈ When the Uds Transport Protocol implementation has a final feedback of the network layer, whether the UDS message triggered for transmission (see [SWS_DM_00348]) could be sent on the network or not, it shall notify DM by calling TransmitConfirmation ([SWS_DM_00312]) on its UdsTransportProtocolMgr reference ((see [SWS_DM_00330]) setting the message argument to the message parameter of the Transmit call ([SWS_DM_00348]). ⌋*(SRS_Diag_04168)*

[SWS_DM_00351] Confirmation Result ⌈ When the the network layer was able to send the UDS response message to the network, the result argument in the TransmitConfirmation shall be set to kTransmitOk, otherwise to kTransmitFailed. ⌋*(SRS_Diag_04168)*

7.1.1.4 Channel Notifications

Each incoming UDS request message is assigned an exact Uds Transport Protocol implementation specific Channel. With the normal request/reply paradigm in diagnostics, the UDS response message is sent out at the same Channel, from

which the UDS request has been received. Therefore the Channel identifier is given to the DM in IndicateMessage (see [SWS_DM_00309]) in the form of parameter global_channel_id. The Channel part from this parameter is then used in the corresponding response in Transmit (see [SWS_DM_00327]).

There are use cases, where a diagnostic request might be answered deferred after the restart of the DM. The UDS service for ECU reset is a candidate for such a requirement. The upcoming requirements shall cover this use case.

[SWS_DM_00356] Requesting Notification of a channel reestablishment ⌈ The DM shall call the NotifyReestablishment (see [SWS_DM_00326]) method of a Uds Transport Protocol implementation, with the parameter channel_id set to the identifier of the Channel, where it needs a re-establishment notification. ⌋ *(SRS_Diag_04168)*

[SWS_DM_00357] Validity/lifetime of a Notification Request ⌈ A notification request registered at a Uds Transport Protocol implementation according to [SWS_DM_00356] is valid only for the next call to Start until the following call to Stop of this Uds Transport Protocol implementation. ⌋*(SRS_Diag_04168)*

[SWS_DM_00358] Notification of a channel reestablishment ⌈ Uds Transport Protocol implementation shall call ChannelReestablished on its UdsTransportProtocolMgr reference ((see [SWS_DM_00330]) setting the global_channel_id argument to the tuple consisting of its own handler_id and the ChannelID it has received in NotifyReestablishment (see [SWS_DM_00356]) once, in case it detects, that the underlying network Channel represented by ChannelID is getting available again. ⌋*(SRS_Diag_04168)*

[SWS_DM_00359] Persistent Storage of Notification Request ⌈ Uds Transport Protocol implementation shall store the notification request (see [SWS_DM_00356]) persistently, to be able to fulfill the notification even after a DM restart. ⌋ *(SRS_Diag_04168)*

7.1.2 DoIP

[SWS_DM_00005] DoIP Support ⌈ DM shall implement/provide a UDS Transport Layer implementation on Ethernet compliant with ISO-13400[4], also called DoIP. ⌋ *(SRS_Eth_00083)*

[SWS_DM_00475] DoIP Version ⌈ DM shall support following version of the DoIP ISO 13400-2 specification: 2012. ⌋*()*

Note: According to the specification, for the vehicle identification request message 0xFF shall be expected.

[SWS_DM_00449] Supported DoIP message types ⌈ DM shall support following listed DoIP message types: ⌋*(SRS_Eth_00026, SRS_Eth_00080, SRS_Eth_00082, SRS_Eth_00083, SRS_Eth_00084, SRS_Eth_00027)*

Payload type value	Payload type Name
0x0000	Generic DoIP header negative acknowledge
0x0001	Vehicle identification
0x0002	Vehicle identification request message with EID
0x0003	Vehicle identification request message with VIN
0x0004	Vehicle announcement message/vehicle identification response message
0x0005	Routing activation request
0x0006	Routing activation response
0x0007	Alive check request
0x0008	Alive check response
0x4001	DoIP entity status request
0x4002	DoIP entity status response
0x4003	Diagnostic power mode information request
0x4004	Diagnostic power mode information response
0x8001	Diagnostic message
0x8002	Diagnostic message positive acknowledgement
0x8003	Diagnostic message negative acknowledgement

[SWS_DM_00205] Providing the VIN in DoIP protocol messages ⌈ If the DM needs to know VIN to be able to react or answer on some DoIP messages, it shall obtain it by using the method Read of the service interface VINInformation. ⌋ *(SRS_Eth_00026)*

[SWS_DM_00434] Providing the PowerMode in DoIP protocol messages ⌈ If the DM needs to know the PowerMode to be able to react or answer on any DoIP message, it shall obtain it by reading the value of the field PowerMode of the service interface DoIPPowerModeInformation. ⌋*(SRS_Eth_00026)*

[SWS_DM_00436] Providing the GID in DoIP protocol messages ⌈ If the DM needs to know the GID and the status of the GID to be able to react or answer on any DoIP message, it shall obtain it by reading the value of the field GID of the service interface DoIPGroupIdentification. ⌋*()*

7.1.3 Dispatching of UDS Requests

[SWS_DM_00390] Dispatching physical Request ⌈ DM shall dispatch each UDS physical request to the Diagnostic Server instance responsible for the SoftwareCluster with diagnosticAddress matching the TargetAddress of the received UDS request and addressSemantics set to physicalAddress. ⌋ *(SRS_Diag_04216)*

[SWS_DM_00391] Dispatching functional Request ⌈ DM shall dispatch each UDS functional request to all Diagnostic Server instances responsible for those SoftwareClusters with a diagnosticAddress matching the TargetAddress of the received UDS request and addressSemantics set to functionalAddress. ⌋ *(SRS_Diag_04216)*

7.2 Diagnostic Server

The AUTOSAR adaptive platform is able to be extended with new software packages without re-flashing the entire ECU. The individual software packages are described by SoftwareClusters. To support the current approaches of diagnostic management (like software updates), each SoftwareCluster has its own diagnosticAddresses. For details on the semantics and precise configuration of SoftwareClusters, see [12].

DM is intended to support an own Diagnostic Server instance per installed SoftwareCluster. All Diagnostic Server instances share the same UDS Transport-Layer (see Figure 7.1) and each Diagnostic Server manages its own resources.

[SWS_DM_00420] Instantiation of Diagnostic Server ⌈ DM shall instantiate an independent Diagnostic Server per configured SoftwareCluster which references a DiagnosticContributionSet in the role of diagnosticExtract with dedicated resources and functionality configured by this DiagnosticContributionSet. ⌋*(SRS_Diag_04216)*

Details on required configuration items are described in section 7.2.3.

This chapter focuses on requirements concerning a single Diagnostic Server, hence we assume that

- requests from Diagnostic Clients are already dispatched towards this Diagnostic Server according to [SWS_DM_00390] and [SWS_DM_00391],
- DEXT configuration elements used in a requirement are meant to be part of the DiagnosticContributionSet associated to the Diagnostic Server according to [SWS_DM_00420].

In particular, we note that requests addressing different SoftwareClusters shall be processed independently by the respective Diagnostic Servers.

7.2.1 Diagnostic Communication Management

A central element in the handling of diagnostic communication is the term Diagnostic Conversation, which is described in section 7.2.1.1. A UDS request is always processed in the context of a Diagnostic Conversation. A single Diagnostic Server can handle multiple Diagnostic Conversations in parallel. In contrast to Classic Platform, Adaptive Platform provides two different modes of parallelism: fully and pseudo parallel mode.

7.2.1.1 Diagnostic Conversations

A Diagnostic Conversation depicts a conversation between a distinct Diagnostic Client and a Diagnostic Server. In contrast to CP, on AP the details of connections

between Diagnostic Clients and Diagnostic Servers are not statically configured, but a Diagnostic Conversation is dynamically allocated during run-time of the Diagnostic Server.

For an incoming UDS request, the Diagnostic Server is identified via the target address of the UDS request (see [SWS_DM_00390], [SWS_DM_00391]), whereas the identification of the Diagnostic Client is transport layer specific.

[SWS_DM_00421] Identification of a Diagnostic Client ⌈ The Diagnostic Server shall identify a Diagnostic Client by means of the tuple of source_addr and gobal_channel_id provided by the TP Layer on call of IndicateMessage, see [SWS_DM_00347]. ⌋*(SRS_Diag_04005)*

[SWS_DM_00046] Each Diagnostic Conversation has its own session resources ⌈ The Diagnostic Server shall provide each Diagnostic Conversation with its own and independently managed diagnostic session, which can be any valid UDS session type. ⌋*(SRS_Diag_04119, SRS_Diag_04006)*

[SWS_DM_00047] Each Diagnostic Conversation has its own security-level resources ⌈ The Diagnostic Server shall provide each Diagnostic Conversation with its own and independently managed security-level. ⌋*(SRS_Diag_04005)*

7.2.1.1.1 Parallel Client Handling Variants

There are generally various approaches for a server (which the Diagnostic Server implements) how to handle parallel/concurrent client requests. The ISO 14229-1[1] does not prescribe a certain approach, because different variants of parallelism also require different amount of resources available within an ECU. Since the ISO 14229-1 also needs to support ECUs which are low on resources, it allows for greater flexibility in terms of supported parallelism.

[SWS_DM_00016] Configurable number of supported parallel Diagnostic Conversations ⌈ Diagnostic Server shall provide a configuration parameter, how many parallel Diagnostic Conversations (resp. Diagnostic Clients) it shall support. ⌋*(SRS_Diag_04166)*

[SWS_DM_00011] Selectability of parallelism mode ⌈ DM shall allow, that it can be configured, whether Diagnostic Server supports fully parallel client mode or pseudo parallel client mode. ⌋*(SRS_Diag_04166, SRS_Diag_04209, SRS_Diag_04210)*

Pseudo Parallel Mode The characteristic of this parallelism mode is, that there is only a real parallelism as long as no Diagnostic Client switches to a non-default session. At the point in time one Diagnostic Client has switched to a non-default session, requests of other diagnostic clients (other Diagnostic Conversations) get rejected with the exception if the newly requested Diagnostic Conversation has a higher priority than the current Diagnostic Conversation in non-default session. This characteristic of the 'pseudo parallel mode' means, that

the diagnostic session state is not an individual state per Diagnostic Client, but it becomes a **global state for the entire Diagnostic Server**.

Fully Parallel Mode The characteristic of this parallelism mode is, that it more reflects the classical client-server architectures from the business IT, where a great extent of parallelism is provided by the server and where each client has its own conversational context with the server, totally shielded from other clients. The session context is also well known from web based technology, where it is naturally/common sense, that it is a separate state/context individually for each client. This Fully Parallel Mode obviously requires more resources from the ECU (Diagnostic Server) acting as the server compared to the Pseudo Parallel Mode. This is an important reason, that the ISO did not require it from UDS ISO 14229-1[1] compliant ECUs as default implementation for handling of parallel clients. Previous ECUs (i.e. based on the CP) were not always capable of providing this. AP based ECUs are not resource-restricted in the same way, so the implementation of Fully Parallel Mode is usually possible.

A Diagnostic Server configured for Fully Parallel Mode allows, that it has at the same time N Diagnostic Conversations) with N different Diagnostic Clients, where each is in a — possibly different — non-default session.

The different behavior of the Diagnostic Server depending on the configured parallelism mode is enforced via specification items that distinguish on the parallelism mode of the Diagnostic Server. This applies to

- the evaluation of incoming UDS requests as described in section 7.2.1.2,
- processing of UDS requests for UDS Services SessionControl (0x10).

In addition, note that some UDS Services involve global aspects of the Diagnostic Server, e.g. the ControlDTCSetting Service 0x85, that cannot be handled independently on multiple Diagnostic Conversations. Such UDS Services require additional restrictions to avoid or coordinate parallel execution. Detailed specification of such restrictions is given per UDS Service in section 7.2.1.6, if applicable.

7.2.1.1.2 Life-cycle of a Diagnostic Conversation

The life-cycle of a Diagnostic Conversation starts with the first reception of a UDS request from the given Diagnostic Client to the Diagnostic Server and ends either if it is canceled (see section 7.2.1.7) or if **all** of the following conditions are satisfied:

- UDS request processing is finished by either
  - sending positive or final negative response and processing `TransmitConfirmation` call from TP-layer according to [SWS_DM_00350],
  - suppressing positive response according to [SWS_DM_00365],

  - suppressing negative response according to [SWS_DM_00366].
  - suppressing any response according to [SWS_DM_00367].
- associated Session is the Default Session.

Note: A Diagnostic Conversation in Non-Default Session is kept alive, as long as no Session time-out occurred. In this case, possibly multiple UDS requests are processed within this Lifecycle.

7.2.1.1.3 Diagnostic Conversation Service Interface

In some cases, the current state of a Diagnostic Conversation needs to be known by some Adaptive Applications. For this purpose, the Diagnostic Server provides instances of the Service Interface DiagnosticConversation.

[SWS_DM_00422] Instantiation of Diagnostic Conversation Service Interface ⌈ The Diagnostic Server shall provide as many instances of the Service Interface DiagnosticConversation as the number of potential parallel Diagnostic Clients is configured according to [SWS_DM_00016]. ⌋*(SRS_Diag_04166)*

[SWS_DM_00423] Assignment of Diagnostic Conversation to Service Instances ⌈ On establishment of a new Diagnostic Conversation, the Diagnostic Server shall assign this Diagnostic Conversation to an inactive DiagnosticConversation Service Instance, i.e. the the field value of ActivityStatus is set to kInactive. After assignment, the fields of the DiagnosticConversation Service Instance shall be updated according to the state of the given Diagnostic Conversation, i.e.,

- ActivityStatus set to kActive,
- Identifier matching the values of IndicateMessage call that initiated the creation of this Diagnostic Conversation (see [SWS_DM_00347]),
- DiagnosticSession matching the Diagnostic Session of this Diagnostic Conversation,
- DiagnosticSecurityLevel matching the Diagnostic Security Level of this Diagnostic Conversation.

⌋*(SRS_Diag_04166)*

[SWS_DM_00484] Updating DiagnosticConversation Service Instance fields ⌈ During the life-cycle of a Diagnostic Conversation, the Diagnostic Server shall update the fields of the assigned DiagnosticConversation according to any change of the state of the Diagnostic Conversation. ⌋*(SRS_Diag_04166)*

[SWS_DM_00424] Reset Service Instance fields on end of Diagnostic Conversation ⌈ If the life-cycle of a Diagnostic Conversation ends, the Diagnostic

Server shall reset the field values of the assigned DiagnosticConversation Service Instance to its predefined initial values. ⌋(*SRS_Diag_04166*)

Besides the described informative character of the DiagnosticConversation Service Interface, it also provides methods for interaction with the state of a Diagnostic Conversation.

[SWS_DM_00435] Default session change trigger from AAs ⌈ If ResetToDefaultSession method is called on a DiagnosticConversation Service Instance with assigned Diagnostic Conversation, the Diagnostic Server shall complete the latest ongoing request and then switch the Diagnostic Session of this Diagnostic Conversation to Default Session. ⌋(*SRS_Diag_04006*)

[SWS_DM_00483] Cancellation trigger from AAs ⌈ If Cancel method is called on a DiagnosticConversation Service Instance with assigned Diagnostic Conversation, the Diagnostic Server shall cancel this Diagnostic Conversation according to [SWS_DM_00482]. ⌋(*SRS_Diag_04167*)

7.2.1.2 Assignment of UDS requests to Diagnostic Conversations

A UDS request is always processed within the context of a Diagnostic Conversation. On reception, the Diagnostic Server has to choose from the following three options:

- assign the UDS request to an existing Diagnostic Conversation,
- establish a new Diagnostic Conversation and assign the UDS request to this Diagnostic Conversation,
- reject the UDS request.

The evaluation which option to choose involves several steps that are summarized in Figure 7.2. The following requirements provide the details.

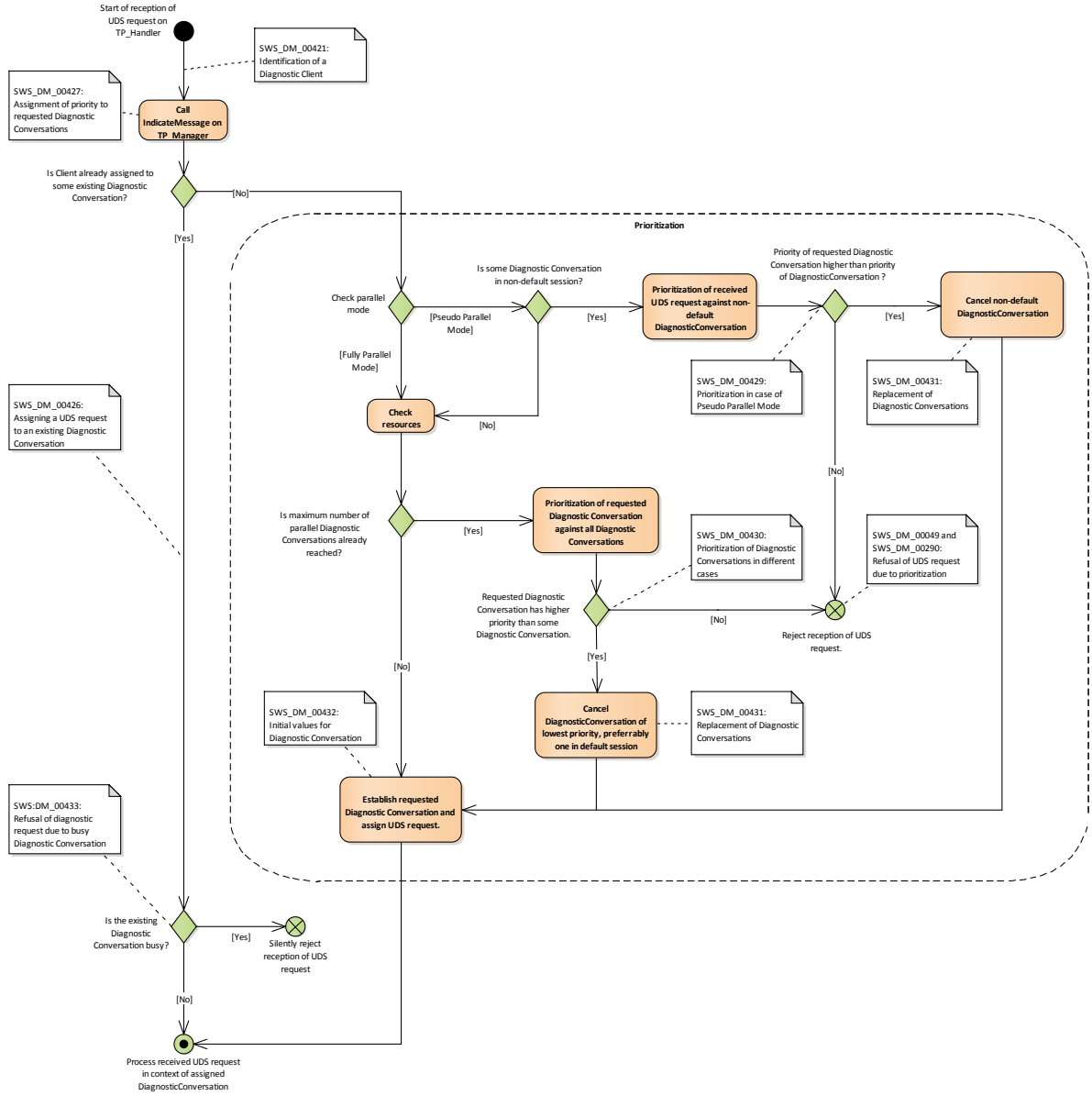

Figure 7.2: UDS request assignment to a Diagnostic Conversation and Prioritization

[SWS_DM_00425] Procedure to assign UDS requests to Diagnostic Conversations ⌈ The Diagnostic Server shall handle a newly received UDS request as specified in Figure 7.2. ⌋*(SRS_Diag_04166)*

[SWS_DM_00426] Assigning a UDS request to an existing Diagnostic Conversation ⌈ If a UDS request is received and there already exists a Diagnostic Conversation associated to the transmitting Diagnostic Client, then the Diagnostic Server shall assign this UDS request to the same Diagnostic Conversation. ⌋*(SRS_Diag_04166)*

Note that the assignment of a UDS request to a Diagnostic Conversation does not necessarily mean that the UDS request is actually processed, see [SWS_DM_00433].

7.2.1.2.1 Prioritization

If the Diagnostic Server lacks resources for new Diagnostic Conversations, a prioritization of the requested Diagnostic Conversation against existing Diagnostic Conversations shall take place. For a Diagnostic Server in pseudo parallel mode, prioritization is also required in case of an existing Diagnostic Conversation in non-default session.

[SWS_DM_00427] Priority of a Diagnostic Conversation ⌈ The Diagnostic Server shall take as the priority of a Diagnostic Conversation the respective value provided by IndicateMessage call according to [SWS_DM_00309] that established the given Diagnostic Conversation. ⌋*(SRS_Diag_04005)*

[SWS_DM_00428] Treatment of priority values ⌈ The Diagnostic Server shall consider a lower value as higher priority and vice versa. In particular, priority value 0 represents highest priority. ⌋*(SRS_Diag_04005)*

[SWS_DM_00429] Prioritization in case of Pseudo Parallel Mode and active non-default session ⌈ If the Diagnostic Server is configured for Pseudo Parallel Mode according to [SWS_DM_00011], the Diagnostic Server shall check if a Diagnostic Conversation is in non-default session. If both is the case, the Diagnostic Server shall compare the priority of the requested Diagnostic Conversation against the priority of the given Diagnostic Conversation in non-default Session: If the priority of the requested Diagnostic Conversation is higher than the priority of the Diagnostic Conversation in non-default Session, the Diagnostic Server shall replace the Diagnostic Conversation in non-default Session by the requested Diagnostic Conversation according to [SWS_DM_00431] and assign the UDS request to the newly established Diagnostic Conversation. ⌋ *(SRS_Diag_04005)*

[SWS_DM_00430] Prioritization against all Diagnostic Conversations ⌈ On prioritization, the Diagnostic Server shall compare the priority of the requested Diagnostic Conversation against the priorities of the existing Diagnostic Conversations:

- If all priorities of the existing Diagnostic Conversations are higher or equal to the priority of the requested Diagnostic Conversation, the Diagnostic Server shall refuse the UDS request according to [SWS_DM_00049] and [SWS_DM_00290].
- If some priority of the existing Diagnostic Conversations is lower than the priority of the requested Diagnostic Conversation, the Diagnostic Server shall replace the Diagnostic Conversation of lowest priority by the requested Diagnostic Conversation according to [SWS_DM_00431] and assign the UDS request to the newly established Diagnostic Conversation.

If multiple Diagnostic Conversations exist with the same lowest priority, the Diagnostic Server shall prefer replacement of a Diagnostic Conversation within default Session before replacement of a Diagnostic Conversation in non-default Session.

⌋(*SRS_Diag_04005*)

7.2.1.2.2 Replacement of Diagnostic Conversations and initial values

[SWS_DM_00431] Replacement of Diagnostic Conversations ⌈ On replacement of a given Diagnostic Conversation by a requested Diagnostic Conversation, the Diagnostic Server shall cancel the given Diagnostic Conversation according to [SWS_DM_00482] and establish a new Diagnostic Conversation as requested. ⌋(*SRS_Diag_04167*)

[SWS_DM_00432] Initial values for Diagnostic Conversation ⌈ For a newly established Diagnostic Conversation, the Diagnostic Server shall use the following initial values:

- Session set to Default Session,
- Security Level set to status Locked.

⌋(*SRS_Diag_04166*)

7.2.1.2.3 Refusal of incoming diagnostic request

[SWS_DM_00433] Refusal of diagnostic request due to busy Diagnostic Conversation ⌈ If a UDS request is assigned to a Diagnostic Conversation that has not finished processing of a formerly assigned UDS request, then the Diagnostic Server shall ignore the new UDS request according to [SWS_DM_00386]. ⌋(*SRS_Diag_04020*)

[SWS_DM_00049] Refusal of diagnostic request due to prioritization with BusyRepeatRequest ⌈ If prioritization demands refusal of an incoming UDS request and the configuration parameter DiagnosticCommonProps.responseOnSecondDeclinedRequest is TRUE, the Diagnostic Server shall accept this request according to [SWS_DM_00385] without further processing and a negative response with NRC 0x21 (BusyRepeatRequest) shall be issued for this request. ⌋(*SRS_Diag_04167*)

[SWS_DM_00290] Refusal of diagnostic request due to prioritization without response ⌈ If prioritization demands refusal of an incoming UDS request and the configuration parameter DiagnosticCommonProps.responseOnSecondDeclinedRequest is FALSE, the Diagnostic Server shall ignore this request according to [SWS_DM_00386] without further processing and no response shall be issued. ⌋(*SRS_Diag_04167*)

7.2.1.3 UDS request Validation/Verification

[SWS_DM_00096] Validation Steps and Order ⌈ The Diagnostic Server shall execute the request validation, negative response code determination and processing according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196, SRS_Diag_04203)*

ISO 14229-1[1] describes a common processing for all requests in "Figure 5 – General server response behavior". There are further optional SID specific processing sequences. This document describes the Diagnostic Server behavior for certain types of checks:

- **manufacturer specific failure detected?** Decision by applying manufacturer specific checks according to section 7.2.1.3.4
- **SID supported?** Decision according to section 7.2.1.3.2
- **SID supported in active session?** Decision according to section 7.2.1.3.3
- **SID security check o.k.?** Decision according to section 7.2.1.3.3
- **supplier-specific failure detected?** Decision by applying supplier-specific checks according to section 7.2.1.3.4

[SWS_DM_00097] Abort on failed verification step ⌈ Whenever one of the verification steps fails, further processing of the request shall be aborted and a negative response shall be sent back. ⌋*(SRS_Diag_04196)*

The negative response code to be used will be defined in each step described in the following sections.

7.2.1.3.1 UDS request format checks

[SWS_DM_00098] UDS message checks ⌈ The Diagnostic Server shall check, whether the diagnostic request is syntactically correct. I.e. whether it conforms to ISO 14229-1 message format specification. If it does not conform, the Verification shall be considered as failed and the negative response code shall be 0x13 (incorrectMessageLengthOrInvalidFormat) ⌋*(SRS_Diag_04203)*

7.2.1.3.2 Supported service checks

[SWS_DM_00099] Supported Service SID level checks ⌈ The Diagnostic Server shall check, whether there is a configured internal or external service processor for the incoming diagnostic request. If there is no service processor on SID level, the Verification shall be considered as failed and the negative response code shall be 0x11 (serviceNotSupported) ⌋*(SRS_Diag_04203)*

[SWS_DM_00100] Supported Service subfunction level checks ⌈ The Diagnostic Server shall check, whether there is a configured internal or external service

processor for the incoming diagnostic request. If there exists a service processor on SID level, but not for the subfunction of the request, the Verification shall be considered as failed and the negative response code shall be 0x12 (subFunctionNotSupported) ⌋ *(SRS_Diag_04203)*

7.2.1.3.3 Session and Security Checks

[SWS_DM_00101] Session Access SID level Permission ⌈ The Diagnostic Server shall check, whether the service processor (DiagnosticServiceInstance), which is assigned to handle the service has the permission to process the service in the current Diagnostic Session according to its DiagnosticAccessPermission.diagnosticSession. If DiagnosticServiceInstance has no access permissions in the current Diagnostic Session and:

- either the SID of the service has no subfunction
- or all other sub-functions also have no access permissions in the current Diagnostic Session,

the Verification shall be considered as failed and the negative response code shall be 0x7F (serviceNotSupportedInActiveSession) ⌋*(SRS_Diag_04203, SRS_Diag_04006)*

[SWS_DM_00102] Session Access subfunction level Permission ⌈ The Diagnostic Server shall check, whether the service processor (DiagnosticServiceInstance), which is assigned to handle the service has the permission to process the service in the current Diagnostic Session according to its DiagnosticAccessPermission.diagnosticSession. If DiagnosticServiceInstance has no access permissions in the current Diagnostic Session and:

- the SID of the service has subfunctions
- and at least one other sub-functions has access permissions in the current Diagnostic Session,

the Verification shall be considered as failed and the negative response code shall be 0x7E (subFunctionNotSupportedInActiveSession) ⌋*(SRS_Diag_04203, SRS_Diag_04006)*

[SWS_DM_00103] Security Access level Permission ⌈ The Diagnostic Server shall check, whether the service processor (DiagnosticServiceInstance), which is assigned to handle the service has the permission to process the service in the current Security-Level according to its DiagnosticAccessPermission.securityLevel. If DiagnosticServiceInstance has no access permissions in the current Security-Level, the Verification shall be considered as failed and the negative response code shall be 0x33 (securityAccessDenied). ⌋*(SRS_Diag_04203, SRS_Diag_04005)*

7.2.1.3.4 Manufacturer and Supplier Permission Checks and Confirmation

[SWS_DM_00106] Signature of Manufacturer Permission Check Method ⌈ The Diagnostic Server shall call the method Validate on each received request message. In case a call returned an ApplicationError contained in the ApplicationErrorSet UDSNegativeResponseCode, the Verification shall be considered as failed and the negative response code shall be equal to the value of the ApplicationError provided by UDSNegativeResponseCode. ⌋*(SRS_Diag_04199)*

[SWS_DM_00108] Signature of Supplier Permission Check Method ⌈ The Diagnostic Server shall call the method Validate on each received request message. In case a call returned an ApplicationError contained in the ApplicationErrorSet UDSNegativeResponseCode, the Verification shall be considered as failed and the negative response code shall be equal to the value of the errorContext provided by UDSNegativeResponseCode. ⌋*(SRS_Diag_04199)*

[SWS_DM_00341] Confirmation of service processing ⌈ The Diagnostic Server shall call the method Confirmation on every service instances for which Validate was called. If message handling results in sending a positive or negative response, the Confirmation call shall be deferred after reception of TransmitConfirmation. In any other case, it shall be the last step of request processing. ⌋ *(SRS_Diag_04019, SRS_Diag_04172)*

[SWS_DM_00367] No service processing ⌈ If Manufacturer- or Supplier Permission Check return the ApplicationError kNoProcessingNoResponse, the Diagnostic Server shall call without any service processing the Confirmation with status parameter set to kNoProcessingNoResponse and do no response message. ⌋*(SRS_Diag_04196)*

7.2.1.3.5 Condition checks

In some cases, diagnostic functionality shall only be executed if the vehicle is in a certain state. An example is the condition is that the vehicle is stopped (vehicle speed == 0).

[SWS_DM_00111] Configurable environment condition checks ⌈ The Diagnostic Server shall perform a condition check when the ISO 14229-1[1] mentions a service specific “Condition check” in the defined NRC handling for a given diagnostic service. The Diagnostic Server shall send the configured NRC value (see [SWS_DM_00289]) if the condition is not fulfilled. ⌋*(SRS_Diag_04199)*

[SWS_DM_00112] Condition check definition ⌈ The Diagnostic Server shall execute a condition check according to [SWS_DM_00111] by the presence of a DiagnosticEnvironmentalCondition referenced in the role environmentalCondition by the processed DiagnosticServiceInstance. ⌋*(SRS_Diag_04199)*

[SWS_DM_00286] Configurable environmental condition check execution ⌈ The Diagnostic Server shall execute an environmental condition check before executing the requested service if defined. (see DiagnosticEnvironmentalCondition element from DEXT [2]). ⌋*(SRS_Diag_04199)*

[SWS_DM_00287] Configurable environmental condition check criteria ⌈ The environmental condition check shall be done by evaluation of the configured DiagnosticEnvConditionFormula. ⌋*(SRS_Diag_04199)*

The DiagnosticEnvConditionFormula may reference a DiagnosticDataElement by a DiagnosticEnvDataCondition with a logical operator given as DiagnosticEnvCompareCondition.

[SWS_DM_00288] Configurable environmental condition check evaluates to TRUE ⌈ If the computation of the DiagnosticEnvConditionFormula evaluated to TRUE, the Diagnostic Server shall execute the requested service. ⌋*(SRS_Diag_04199)*

[SWS_DM_00289] Configurable environmental condition check evaluates to FALSE ⌈ The Diagnostic Server shall send the NRC defined in nrcValue, if the computation of the DiagnosticEnvConditionFormula evaluated to FALSE. If nrcValue does not define a NRC, the Diagnostic Server shall send NRC 0x22 (ConditionsNotCorrect). ⌋*(SRS_Diag_04199)*

7.2.1.4 UDS response handling

7.2.1.4.1 Positive and negative responses

[SWS_DM_00376] Positive response processing ⌈ If an external service processor did not raise an ApplicationError, the Diagnostic Server shall return a positive response. ⌋*(SRS_Diag_04196)*

[SWS_DM_00364] Negative response processing ⌈ If the external processor raised an ApplicationError contained in the ApplicationErrorSet UDSNegativeResponseCode, the Diagnostic Server shall return a negative response with the value of the errorCode of the raised ApplicationError. For details see ISO 14229-1[1]; chapter 10.2. ⌋*(SRS_Diag_04196)*

7.2.1.4.2 Suppression of responses

[SWS_DM_00365] Suppression of positive response in accordance to ISO 14229-1[1] ⌈ In the case that the "suppressPosRspMsgIndicationBit" is set in the request, the Diagnostic Server shall suppress the positive response. ⌋*(SRS_Diag_04020)*

[SWS_DM_00366] Suppression of negative response for functional requests in accordance to ISO 14229-1[1] ⌈ If the external processor raised an ApplicationError contained in the ApplicationErrorSet UDSNegativeResponseCode, the

Diagnostic Server shall suppress a negative response for the following errorCodes:

- kServiceNotSupported,
- kSubfunctionNotSupported,
- kRequestOutOfRange,
- kServiceNotSupportedInActiveSession or
- kSubFunctionNotSupportedInActiveSession

and the request is functional addressed. ⌋(*SRS_Diag_04020*)

7.2.1.4.3 Sending busy Responses

[SWS_DM_00368] Sending busy responses ⌈ If the Diagnostic Server is able to perform a diagnostic service, but needs additional time to finish the task and prepare the response, then the Diagnostic Server shall send a negative response with NRC 0x78 (Response pending) when reaching the response time (p2ServerMax/p2StarServerMax). ⌋(*SRS_Diag_04016*)

[SWS_DM_00369] Maximum number of busy responses ⌈ If the number of negative responses for a requested diagnostic request reaches the value defined in the configuration parameter maxNumberOfRequestCorrectlyReceivedResponsePending, the Diagnostic Server module shall cancel the processing the active diagnostic request (according to [SWS_DM_00482]) and send a negative response with NRC 0x10 (General reject). ⌋(*SRS_Diag_04016*)

7.2.1.5 Keep track of active non-default sessions

[SWS_DM_00380] Support for S3 timer ⌈ The Diagnostic Server shall provide support for $S3_{Server}$ (session timeout) with a fixed value of 5 second. The timer handling shall be implemented according to ISO 14229-2[13]. ⌋(*SRS_Diag_04006*)

[SWS_DM_00381] Session timeout ⌈ Whenever a non-default session is active and when the session timeout ($S3_{Server}$) is reached without receiving any diagnostic request, the Diagnostic Server shall reset to the default session state. Diagnostic Server internal states for service processing shall be reset according to ISO 14229-2[13]. ⌋(*SRS_Diag_04006*)

[SWS_DM_00382] Session timeout start ⌈ The session timeout timer ($S3_{server}$) shall be started on

- Completion of any final response message or an error indication during sending of the response ([SWS_DM_00312]/TransmitConfirmation)

- Completion of the requested action in case no response message (positive and negative) is required / allowed.
- In case of an error during the reception of a multi-frame request message ([SWS_DM_00310]/ `NotifyMessageFailure`)

Start of $S3_{Server}$ means reset the timer and start counting from the beginning. ⌋ *(SRS_Diag_04006)*

[SWS_DM_00383] Session timeout stop ⌈ The session timeout timer ($S3_{Server}$) shall be stopped when the reception of an `UDS` message was indicated ([SWS_DM_00309]/`IndicateMessage`). ⌋*(SRS_Diag_04006)*

7.2.1.6 UDS service processing

This chapter describes the UDS service processing behavior of the `Diagnostic Server`.

[SWS_DM_00127] Availability of diagnostic service processors ⌈ The `Diagnostic Server` shall provide a service processor on SID level for all services by existence of a `DiagnosticServiceClass` referenced by a `DiagnosticServiceInstance.serviceClass`. ⌋*(SRS_Diag_04196)*

7.2.1.6.1 Supported UDS Services

[SWS_DM_00104] Supported UDS Services ⌈ The `Diagnostic Server` shall support the following listed UDS services:

SID	Service	Support Type
0x10	DiagnosticSessionControl	Internally
0x11	ECUReset	Externally
0x14	ClearDiagnosticInformation	Internally
0x19	ReadDTCInformation	Internally
0x22	ReadDataByIdentifier	Internally & Externally
0x27	SecurityAccess	Internally & Externally
0x28	CommunicationControl	Externally
0x2E	WriteDataByIdentifier	Externally
0x31	RoutineControl	Externally
0x34	RequestDownload	Externally
0x35	RequestUpload	Externally
0x36	TransferData	Externally
0x37	RequestTransferExit	Externally
0x3E	TesterPresent	Internally
0x85	ControlDTCSetting	Internally
0x86	ResponseOnEvent	Internally

⌋*(SRS_Diag_04196, SRS_Diag_04180, SRS_Diag_04198)*

Note: Support Type `Internally` means, that the service with the given SID can be completely processed internally within the `Diagnostic Server` without relying on external functionality - typically in form of an `AA`. Support Type `Externally` means, that the `Diagnostic Server` needs to call an external function, to be able to process the service with the given SID. The mixed support Type `Internally & Externally` means, that for the service with the given SID partially calls to an external function have to be done, but it partially could be also handled internally.

7.2.1.6.2 Common service processing items

This chapter contains rules for service processors, share among multiple services.

Memory related UDS services (such as 0x34 RequestDownload) use the request parameter addressAndLengthFormatIdentifier to identify the number of bytes transmitted on the bus for memory address and size. Regardless of the wire representation of address and length information, within the `Diagnostic Server` and external service processors all addresses and data length information are mapped to a uint64 datatype.

[SWS_DM_00129] Supported addressAndLengthFormatIdentifier ⌈ The `Diagnostic Server` shall support for each nibble of the addressAndLengthFormatIdentifier a value between 1 and 8. ⌋*(SRS_Diag_04120)*

[SWS_DM_00130] Not supported addressAndLengthFormatIdentifier ⌈ The `Diagnostic Server` shall send the negative response 0x31 (requestOutOfRange), if an addressAndLengthFormatIdentifier with a value outside the range between 1 and 8 is received. ⌋*(SRS_Diag_04120)*

7.2.1.6.3 Service 0x10 – DiagnosticSessionControl

The UDS service DiagnosticSessionControl is used to enable different diagnostic sessions in the server.

[SWS_DM_00226] Support of UDS service DiagnosticSessionControl ⌈ The `Diagnostic Server` shall provide the UDS service 0x10 DiagnosticSessionControl according to ISO 14229-1[1]. ⌋*(SRS_Diag_04198)*

[SWS_DM_00227] Check for supported sessions ⌈ If the Subfunction addressed by the DiagnosticSessionControl according to [SWS_DM_00226] is not supported by the configuration, i.e., there is no `DiagnosticSession` configured with `id` matching the requested Subfunction value, the `Diagnostic Server` shall return a NRC 0x12 (SubfunctionNotSupported). ⌋*(SRS_Diag_04196)*

In the context of parallel clients, a DiagnosticSessionControl may lead to negative responses even for supported Subfunctions with positive permission checks.

[SWS_DM_00228] Switch to requested Diagnostic Session ⌈ On positive evaluation of a DiagnosticSessionControl request, the `Diagnostic Server` shall switch

the internal representation of Diagnostic Sessions to the DiagnosticSession with id matching the requested Subfunction value, and shall set new timing parameters according to the associated parameters p2ServerMax and p2StarServerMax. ⌋ *(SRS_Diag_04198)*

[SWS_DM_00248] Notification about session change ⌈ If the Diagnostic Server did successfully change the session of a conversation, it shall update the field DiagnosticSession of provided service DiagnosticConversation accordingly. ⌋*(SRS_Diag_04208)*

7.2.1.6.4 Service 0x11 – ECUReset

[SWS_DM_00234] Support of UDS service ECUReset ⌈ The Diagnostic Server shall provide the UDS service 0x11 ECUReset according to ISO 14229-1[1]. ⌋ *(SRS_Diag_04196)*

[SWS_DM_00235] ECUReset service processing ⌈ The Diagnostic Server shall call the method RequestRestart of the interface RequestRestart to process an ECU-Reset. The RestartType parameter shall be set according to the value of the DiagnosticEcuReset.category.

The ExecutionType parameter shall be set:

In case the parameter DiagnosticEcuResetClass.respondToReset is either not present or present and set to DiagnosticResponseToEcuResetEnum.respondBeforeReset to: kImmediate

In case the parameter DiagnosticEcuResetClass.respondToReset is present and set to DiagnosticResponseToEcuResetEnum.respondAfterReset to: kDefered ⌋*(SRS_Diag_04196)*

[SWS_DM_00268] EcuReset positive response processing before reset ⌈ If the external processor did NOT raise an ApplicationError, the Diagnostic Server shall return a positive response before the actual reset, in case the parameter DiagnosticEcuResetClass.respondToReset is either not present or present and set to DiagnosticResponseToEcuResetEnum.respondBeforeReset. ⌋ *(SRS_Diag_04019)*

[SWS_DM_00360] EcuReset positive response processing after reset ⌈ If the external processor did NOT raise an ApplicationError, the Diagnostic Server shall return a positive response after the actual reset if NotifyReestablishment method (see [SWS_DM_00326]) is called (which could also happen after a restart of DM itself), in case the parameter DiagnosticEcuResetClass.respondToReset is present and set to DiagnosticResponseToEcuResetEnum.respondAfterReset. ⌋*(SRS_Diag_04196)*

Note: The information that the reset shall be transmitted after the NotifyReestablishment method (see [SWS_DM_00326]) is called can be stored by a flag in non-volatile memory.

[SWS_DM_00361] EcuReset application error processing ⌈ If RequestRestart raised an ApplicationError contained in the ApplicationErrorSet RequestRestart, the Diagnostic Server shall return a negative response with the value 0x22. ⌋*(SRS_Diag_04196)*

[SWS_DM_00269] Reaction on Unsupported Subfunction ⌈ The Diagnostic Server shall send a negative response 0x12 (SubfunctionNotSupported), if the requested subfunction value is neither in configured range of default subfunction values (requestType, see ISO 14229-1[1]) nor in range of the configured DiagnosticEcuReset.customSubFunctionNumber in the ECU. ⌋*(SRS_Diag_04196)*

7.2.1.6.5 Service 0x14 – ClearDiagnosticInformation

The UDS service ClearDiagnosticInformation is used to clear the ECUs fault memory.

[SWS_DM_00090] Support of UDS service ClearDiagnosticInformation ⌈ The Diagnostic Server shall provide the UDS service 0x14 ClearDiagnosticInformation according to ISO 14229-1[1]. ⌋*(SRS_Diag_04180, SRS_Diag_04196)*

[SWS_DM_00091] Evaluation of ClearDiagnosticInformation parameters ⌈ The Diagnostic Server shall determine the DTC group or single DTC to clear from the 'groupOfDTC' parameter the UDS request. ⌋*(SRS_Diag_04180, SRS_Diag_04117)*

[SWS_DM_00092] Parameter range check for groupOfDTC request parameter ⌈ The Diagnostic Server shall reply with an NRC 0x31 (RequestOutOfRange) if the requested 'groupOfDTC' has no matching configured DTC group according to [SWS_DM_00064] or configured DTC by DiagnosticTroubleCodeUds.udsDtcValue. ⌋*(SRS_Diag_04180, SRS_Diag_04117)*

[SWS_DM_00113] Positive response for UDS service 0x14 ⌈ If Diagnostic Server has cleared the requested 'groupOfDTC', the Diagnostic Server shall send a positive response. ⌋*(SRS_Diag_04196)*

The DTC clearing behavior is described in detail in section 7.2.2.4.6. It consists of resetting the DTC status and deleting snapshot records and extended data records.

[SWS_DM_00114] Limitation to one simultaneous DTC clear operation ⌈ If a DTC clear operation is already in progress, the Diagnostic Server shall deny an UDS request 0x14 and send a negative response 0x22 (conditionsNotCorrect). ⌋ *(SRS_Diag_04196)*

[SWS_DM_00115] Memory error handling while clearing DTCs ⌈ The Diagnostic Server shall return a negative response NRC 0x72 (generalProgrammingFailure) if it encounters an error in the non-volatile memory while clearing the DTCs. ⌋ *(SRS_Diag_04180)*

The definition of a failure of the non-volatile memory is hardware and project specific. In general if the clear DTC operation could not delete the snapshot records, extended

data records and if it could not reset the UDS DTC status byte because the underlying storage system reported and error, a non-volatile memory error can be assumed.

[SWS_DM_00122] UDS response behavior on not allowed clear operations ⌈ If a DTC clear operation is requested and the DTC clear operation shall clear a DTC with a forbidden clear allowance according to [SWS_DM_00481], the `Diagnostic Server` shall send a negative response 0x22 (conditionsNotCorrect) in the following situations:

- it was requested to clear a single DTC and the DTC could not be cleared according to [SWS_DM_00481]
- it was requested to clear a DTC group and all the DTCs of the DTC group could not be cleared according to [SWS_DM_00481]
  (This doesn't apply when one or more DTC are allowed to be cleared.)

⌋*(SRS_Diag_04117)*

[SWS_DM_00159] Allow only to clear `GroupOfAllDTCs` ⌈ If the configuration `DiagnosticCommonProps.clearDtcLimitation` is set to `clearAllDtcs`, the `Diagnostic Server` shall only allow to clear all DTCs via the `GroupOfAllDTC` as defined in [SWS_DM_00065]. In case a different value is given in groupOfDTC request parameter, the `Diagnostic Server` shall return a negative response 0x31 (RequestOutOfRange). ⌋*(SRS_Diag_04117)*

[SWS_DM_00160] Allow to clear single `DTCs` ⌈ If the configuration `DiagnosticCommonProps.clearDtcLimitation` is set to `allSupportedDtcs`, the `Diagnostic Server` shall allow to clear single DTCs or DTCGroups. [SWS_DM_00092] defines the possible and refused values. ⌋*(SRS_Diag_04117)*

[SWS_DM_00162] Point in time for positive response for ClearDTC ⌈ The `Diagnostic Server` shall send a positive response for a ClearDiagnosticInformation service after all memory is cleared in the server. This is regardless how the `Diagnostic Server` memory is organized (splitted, volatile, non-volatile). ⌋*(SRS_Diag_04180, SRS_Diag_04196)*

[SWS_DM_00163] Definition of a failed clear operation with event clear allowed and event combination ⌈ If it is requested to clear a single DTC and multiple `DiagnosticEventToTroubleCodeUdsMapping` referencing this `DiagnosticEventToTroubleCodeUdsMapping.troubleCodeUds` the `Diagnostic Server` shall send a negative response 0x22 (conditionsNotCorrect) if one event forbids the clearance of the DTC according to [SWS_DM_00481]. ⌋*(SRS_Diag_04180)*

[SWS_DM_00164] Definition of a failed clear operation with event clear allowed and clearing a group of DTCs ⌈ If it is requested to clear a group of DTCs, the `Diagnostic Server` shall send a negative response 0x22 (conditionsNotCorrect) if all DTCs of that group of DTC forbid the clearance according to [SWS_DM_00163] or [SWS_DM_00481]. ⌋*(SRS_Diag_04180)*

7.2.1.6.5.1 Clearing user-defined fault memory

According to [SWS_DM_00090] the Diagnostic Server implements an ISO 14229-1[1] compatible UDS service ClearDiagnosticInformation. This implies a limitation that only the primary fault memory can be cleared using this UDS service. To provide means to clear the user-defined fault memories, the Diagnostic Server prospectively implements an agreed proposal by ISO 14229-1 to allow clearance of used defined fault memories. The proposal can be found in the ISO 14229 document: "02_ISO_14229-1_Comments-Summary_2016-09-13.docx". Until the next final release of ISO 14229-1[1] containing this extension, the Diagnostic Server will implement this proposed extension in the way described in this chapter.

The clearance of a user-defined fault memory has the same behavior as the clearing of the primary fault memory. All requirements that are provided to clear the primary fault memory also apply to a clear of a user-defined fault memory. So finally it is a pure extension.

[SWS_DM_00193] Support of a user-defined fault memory clear request ⌈ If the Diagnostic Server receives a a UDS service 0x14 ClearDiagnosticInformation with a length of 5 bytes, the Diagnostic Server shall interpret this request as a request to clear user-defined fault memory. ⌋*(SRS_Diag_04197)*

[SWS_DM_00194] Definition of the user-defined fault memory number for ClearDiagnosticInformation ⌈ If the Diagnostic Server receives a UDS request to clear user-defined fault memory according to [SWS_DM_00193], the DM shall get the number of user-defined fault memory to be cleared from the fifth byte in the request. ⌋ *(SRS_Diag_04197)*

[SWS_DM_00195] Clearing a user-defined memory ⌈ If the Diagnostic Server is requested to clear the user-defined fault memory according to [SWS_DM_00193] and an DiagnosticMemoryDestinationUserDefined.memoryId exists with the requested user-defined memory number according to [SWS_DM_00194], the Diagnostic Server shall clear the requested user-defined fault memory. ⌋ *(SRS_Diag_04197)*

For details about the fault memory clearing process please also refer to section 7.2.2.4.6.

[SWS_DM_00208] Validation of the requested user-defined memory number ⌈ If the Diagnostic Server is requested to clear the user-defined fault memory according to [SWS_DM_00193] and no DiagnosticMemoryDestinationUserDefined.memoryId exists with the requested user-defined memory number according to [SWS_DM_00194], the Diagnostic Server shall return a NRC 0x31 (RequestOutOfRange). ⌋*(SRS_Diag_04197)*

7.2.1.6.6 Service 0x19 – ReadDTCInformation

Some UDS responses for the Service “0x19 – ReadDTCInformation” use the parameter “DTCFormatIdentifier” as part of the response PDU. The Diagnostic Server obtains the value used from the global configuration item DiagnosticCommonProps.typeOfDtcSupported. To provide the correct UDS values, the following mapping is used:

[SWS_DM_00062] Mapping between ISO 14229-1[1] and Autosar Diagnostic Extract Template [2] of the DTCFormatIdentifier ⌈ If a positive response for service 0x19 with the ISO 14229-1[1] parameter “DTCFormatIdentifier” is sent, the Diagnostic Server shall derive the value from DiagnosticCommonProps.typeOfDtcSupported applying the following mapping rule: ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

typeOfDtcSupported	“DTCFormatIdentifier”
iso11992_4	0x03
iso14229_1	0x01
saeJ2012_da	0x00

7.2.1.6.6.1 SF 0x01 – reportNumberOfDTCByStatusMask

[SWS_DM_00244] Support of UDS service ReadDTCInformation, Subfunction 0x01 ⌈ The Diagnostic Server shall support Subfunction 0x01 (reportNumberOfDTCByStatusMask) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category ’REPORT_NUMBER_OF_DTC_BY_STATUS_MASK’. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

[SWS_DM_00061] Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCByStatusMask ⌈ While sending the positive response for ReadDTCInformation.reportNumberOfDTCByStatusMask, the Diagnostic Server shall set the response PDU “DTCFormatIdentifier” according to the mapping of [SWS_DM_00062]. ⌋*(SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.2 SF 0x02 – reportDTCByStatusMask

[SWS_DM_00245] Support of UDS service ReadDTCInformation, Subfunction 0x02 ⌈ The Diagnostic Server shall support Subfunction 0x02 (reportDTCByStatusMask) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category ’REPORT_DTC_BY_STATUS_MASK’. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.3 SF 0x04 – reportDTCSnapshotRecordByDTCNumber

[SWS_DM_00246] Support of UDS service ReadDTCInformation, Subfunction 0x04 ⌈ The Diagnostic Server shall support Subfunction 0x04 (reportDTCSnapshotRecordByDTCNumber) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category 'REPORT_DTC_SNAPSHOT_RECORD_BY_DTC_NUMBER'. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.4 SF 0x06 – reportDTCExtDataRecordByDTCNumber

[SWS_DM_00370] Support of UDS service ReadDTCInformation, Subfunction 0x06 ⌈ The Diagnostic Server shall support Subfunction 0x06 (reportDTCExtDataRecordByDTCNumber) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category 'REPORT_DTC_EXT_DATA_RECORD_BY_DTC_NUMBER'. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.5 SF 0x07 – reportNumberOfDTCBySeverityMaskRecord

[SWS_DM_00247] Support of UDS service ReadDTCInformation, Subfunction 0x07 ⌈ The Diagnostic Server shall support Subfunction 0x07 (reportNumberOfDTCBySeverityMaskRecord) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category 'REPORT_NUMBER_OF_DTC_BY_SEVERITY_MASK_RECORD'. ⌋*(SRS_Diag_04180, SRS_Diag_04157)*

[SWS_DM_00063] Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord ⌈ While sending the positive response for ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord, the Diagnostic Server shall set the response PDU "DTCFormatIdentifier" according to the mapping of [SWS_DM_00062]. ⌋*(SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.6 SF 0x14 – reportDTCFaultDetectionCounter

[SWS_DM_00371] Support of UDS service ReadDTCInformation, Subfunction 0x14 ⌈ The Diagnostic Server shall support Subfunction 0x14 (reportDTCFaultDetectionCounter) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category 'REPORT_DTC_FAULT_DETECTION_COUNTER'. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.7 SF 0x17 – reportUserDefMemoryDTCByStatusMask

[SWS_DM_00372] Support of UDS service ReadDTCInformation, Subfunction 0x17 ⌈ The Diagnostic Server shall support Subfunction 0x17 (reportUserDefMemoryDTCByStatusMask) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category 'REPORT_USER_DEF_MEMORY_DTC_BY_STATUS_MASK'. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.8 SF 0x18 – reportUserDefMemoryDTCSnapshotRecordByDTCNumber

[SWS_DM_00373] Support of UDS service ReadDTCInformation, Subfunction 0x18 ⌈ The Diagnostic Server shall support Subfunction 0x18 (reportUserDefMemoryDTCSnapshotRecordByDTCNumber) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category 'REPORT_USER_DEF_MEMORY_DTC_SNAPSHOT_RECORD_BY_DTC_NUMBER'. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.6.9 SF 0x19 – reportUserDefMemoryDTCExtDataRecordByDTCNumber

[SWS_DM_00374] Support of UDS service ReadDTCInformation, Subfunction 0x19 ⌈ The Diagnostic Server shall support Subfunction 0x19 (reportUserDefMemoryDTCExtDataRecordByDTCNumber) of the UDS service 0x19 ReadDTCInformation according to ISO 14229-1[1], provided the configuration contains a DiagnosticReadDTCInformation of category 'REPORT_USER_DEF_MEMORY_DTC_EXT_DATA_RECORD_BY_DTC_NUMBER'. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.1.6.7 Service 0x22 – ReadDataByIdentifier

The processing of a UDS Service ReadDataByIdentifier (0x22) is described in ISO 14229-1[1], see in particular the evaluation sequence in Figure 15. On processing, the Diagnostic Server needs to perform various checks. The following requirements determine the relation between the input data to be checked and the configuration provided to the Diagnostic Server via DEXT parameters.

[SWS_DM_00170] Realisation of UDS service ReadDataByIdentifier (0x22) ⌈ The Diagnostic Server shall implement the diagnostic service 0x22 ReadDataByIdentifier according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00412] Check requested number of DataIdentifiers ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), the Diagnostic Server shall check the number of the requested DataIdentifiers against the configuration parameter maxDidToRead. ⌋*(SRS_Diag_04203)*

[SWS_DM_00409] Check supported DataIdentifier ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), a requested DataIdentifier shall be considered as supported if and only if there exists a DiagnosticDataIdentifier with id matching the DataIdentifier and this DiagnosticDataIdentifier is referenced by a DiagnosticReadDataByIdentifier. ⌋*(SRS_Diag_04203)*

[SWS_DM_00413] Check supported DataIdentifier in active session ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), a requested DataIdentifier shall be considered as supported in active session if and only if the DataIdentifier is supported according to [SWS_DM_00409] and the active session passes the execution permission check as per [SWS_DM_00101]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00414] Check supported DataIdentifier on active security level ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), a requested DataIdentifier shall be considered as supported on active security level if and only if the DataIdentifier is supported according to [SWS_DM_00409] and the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00408] Retrieving data for requested DataIdentifier ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), the Diagnostic Server shall retrieve the data for a DataIdentifier according to its configuration as described in [SWS_DM_00401], [SWS_DM_00503], [SWS_DM_00504], [SWS_DM_00404], [SWS_DM_00508]. ⌋*(SRS_Diag_04097)*

[SWS_DM_00177] Reaction on ApplicationError ⌈ If the external processor raised an ApplicationError contained in the ApplicationErrorSet UDSNegativeResponseCode, the Diagnostic Server shall return a negative response with the value of the errorCode of the raised ApplicationError. ⌋*(SRS_Diag_04196)*

Note: If multiple DataIdentifer are requested within one ReadDataByIdentifier request, [SWS_DM_00177] might result in a deviation from ISO 14229-1[1] in case the AA raises

an ApplicationError kRequestOutOfRange (resulting in NRC 0x31). According to ISO 14229-1[1], chapter 10.2, a tester expects to receive NRC 0x31 only in case **none** of the requested DataIdentifier are supported. Handling of ApplicationErrors as described in [SWS_DM_00177] might lead to NRC 0x31 on processing one of the requested DataIdentfier without checking the other requested DataIdentifier.

7.2.1.6.8 Service 0x27 – SecurityAccess

[SWS_DM_00236] Realization of UDS service 0x27 SecurityAccess ⌈ The Diagnostic Server shall implement the diagnostic service 0x27 SecurityAccess according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196, SRS_Diag_04005)*

[SWS_DM_00249] Checking Supported Subfunction for RequestSeed ⌈ On reception of a request for UDS Service SecurityAccess (0x27), the Diagnostic Server shall call GetSeed if the requested subfunction value (access type) matches to the value of the instance of DiagnosticSecurityAccess with requestSeedId. The securityAccessDataRecord parameter of the method GetSeed shall be filled with the securityAccessDataRecord provided by the tester. If no data is provided by the tester, the securityAccessDataRecord parameter shall be empty. ⌋*(SRS_Diag_04203)*

Note: The static seed mechanism, as specified in ISO 14229-1[1] - annex I.2 table I.1, needs to be done by the application with the implementation of GetSeed / CompareKey.

[SWS_DM_00507] Length check on UDS Service 0x27 request with Subfunction for RequestSeed ⌈ On reception of a request for UDS Service SecurityAccess (0x27) with subfunction value matching the requestSeedId of a configured DiagnosticSecurityAccess, the Diagnostic Server shall perform the message length check against the optionally configured accessDataRecordSize of the related DiagnosticSecurityLevel. A non-present parameter accessDataRecordSize results in a check against 0 additional request bytes. If the length check fails, the Diagnostic Server shall send NRC 0x13 (IncorrectMessageLengthOrInvalidFormat). ⌋*(SRS_Diag_04203)*

[SWS_DM_00362] Checking Supported Subfunction for CompareKey ⌈ The Diagnostic Server shall call CompareKey when the requested subfunction value (access type) - 1 (to get the corresponding requestSeed) is similar to the value of instance of DiagnosticSecurityAccess with requestSeedId. ⌋*(SRS_Diag_04203)*

[SWS_DM_00363] Unsupported Subfunction ⌈ If the requested subfunction value is not configured (no instances of DiagnosticSecurityAccess with requestSeedId, as well as the corresponding CompareKey values), a negative response 0x12 (SubfunctionNotSupported) shall be returned. (SubFunction not supported). ⌋*(SRS_Diag_04196)*

[SWS_DM_00250] Notification about security-level change ⌈ If Diagnostic Server did successfully change the security-level of a conversation, it shall update the

DiagnosticSecurityLevel field of provided service DiagnosticConversation accordingly. Whether a security level is applicable by the DiagnosticSecurityAccess is defined by securityLevel. ⌋(*SRS_Diag_04208*)

[SWS_DM_00270] Counting of attempts to change security level ⌈ The Diagnostic Server module shall count the number of failed attempts to change a requested security level. The Counter shall be reset if the security level change has passed successfully. ⌋(*SRS_Diag_04208*)

[SWS_DM_00271] Evaluate the number of failed security level change attempts ⌈ The Diagnostic Server shall compare the number of failed DiagnosticSecurityLevel changes with threshold value numFailedSecurityAccess after each failed attempt.
If the number of failed attempts is below the threshold value numFailedSecurityAccess the Diagnostic Server module shall send a negative response with NRC 0x35 (InvalidKey).
If the number of failed attempts reaches the threshold value numFailedSecurityAccess the Diagnostic Server module shall start a delay timer configured with value securityDelayTime (see [SWS_DM_00272]) and send a negative response with NRC 0x36 (exceededNumberOfAttempts).
In both cases a DiagnosticSecurityLevel change must not be done if the attempt failed before. ⌋(*SRS_Diag_04208*)

The delay timer represents the required minimum time between security access attempts, after one time negative response with NRC 0x36 (exceededNumberOfAttempts) was sent out.

[SWS_DM_00272] Expiration of the delay timer ⌈ As long as the delay timer (see [SWS_DM_00271]) configured with threshold value securityDelayTime has not expired, all requests for DiagnosticSecurityLevel change with subfunction value (access type) requestSeed shall be responded with NRC 0x37 (requiredTimeDelayNotExpired).
⌋(*SRS_Diag_04208*)

[SWS_DM_00478] Persistent Storage of failed attempts to change security level ⌈ The Diagnostic Server module shall store the number of failed attempts persistently for every security access type separately. (see [SWS_DM_00270]) ⌋(*SRS_Diag_04208*)

[SWS_DM_00479] Blocking Timer for security access on Restart or Power down - power up cycle ⌈ The Diagnostic Server module shall restart the security delay timer with the higher value of DiagnosticCommonProps.securityDelayTimeOnBoot / DiagnosticSecurityLevel.securityDelayTime of the according DcmDspSecurityRow if at least one of the stored numbers of failed attempts are greater or equal than the threshold value DiagnosticSecurityLevel.numFailedSecurityAccess. The behavior is equal to the behavior on runtime [SWS_DM_00272]) In case failed attempts are lower than the threshold value, the handling is equal to the behavior on runtime. (see [SWS_DM_00270] and [SWS_DM_00271]) ⌋(*SRS_Diag_04208*)

[SWS_DM_00480] Security Access Blocking Timer ⌈ If DiagnosticSecurityAccessClass.sharedTimer exists and is set to true, a shared delay timer instance and shared value DiagnosticSecurityLevel.securityDelayTime shall be used for all security levels. As long as the blocking timer is running and not expired, all requests for every DiagnosticSecurityLevel change with subfunction value (access type) requestSeed shall be responded with NRC 0x37 (requiredTimeDelayNotExpired). (see [SWS_DM_00272]) If DiagnosticSecurityAccessClass.sharedTimer not exists or is set to false, an independent timer instance and timer value shall be used for each security level. ⌋*(SRS_Diag_04208)*

[SWS_DM_CONSTR_00208] Delay time value for sharedTimer ⌈ If DiagnosticSecurityAccessClass.sharedTimer exists and is set to true, the value DiagnosticSecurityLevel.securityDelayTime shall be identical for all configured security levels. ⌋*(SRS_Diag_04208)*

7.2.1.6.9 Service 0x28 – CommunicationControl

[SWS_DM_00140] Realisation of UDS service 0x28 CommunicationControl ⌈ The Diagnostic Server shall implement the diagnostic service 0x28 CommunicationControl according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00252] Reaction on Unsupported Subfunction ⌈ The Diagnostic Server shall check, whether the Subfunction addressed by the CommunicationControl is supported by an existing DiagnosticComControl.category in the configuration and allow further processing. If the Subfunction addressed by the CommunicationControl is not supported by an existing DiagnosticComControl.category in the configuration a negative response 0x12 (SubfunctionNotSupported) shall be returned. ⌋*(SRS_Diag_04203)*

[SWS_DM_00197] Communication control service processing ⌈ The Diagnostic Server shall call the method CommCtrlRequest of the interface CommunicationControl to process a communication control service. ⌋*(SRS_Diag_04169)*

[SWS_DM_00198] Negative Response processing ⌈ If the external processor raised an ApplicationError contained in the ApplicationErrorSet UDSNegativeResponseCode, the Diagnostic Server shall return a negative response with the value of the errorCode of the raised ApplicationError. ⌋*(SRS_Diag_04196)*

[SWS_DM_00199] Positive Response processing ⌈ If the external processor did raise no ApplicationError, the Diagnostic Server shall return a positive response. ⌋*(SRS_Diag_04196)*

7.2.1.6.10 Service 0x2E – WriteDataByIdentifier

The processing of a UDS Service WriteDataByIdentifier (0x2E) is described in ISO 14229-1[1], see in particular the evaluation sequence in Figure 21. On processing, the Diagnostic Server needs to perform various checks. The following requirements determine the relation between the input data to be checked and the configuration provided to the Diagnostic Server via DEXT parameters.

[SWS_DM_00186] Realisation of UDS service WriteDataByIdentifier (0x2E) ⌈ The Diagnostic Server shall implement the diagnostic service 0x2E WriteDataByIdentifier according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00415] Check supported DataIdentifier ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported if and only if there exists a DiagnosticDataIdentifier with id matching the DataIdentifier and this DiagnosticDataIdentifier is referenced by a DiagnosticWriteDataByIdentifier. ⌋*(SRS_Diag_04203)*

[SWS_DM_00416] Check supported DataIdentifier in active session ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported in active session if and only if the DataIdentifier is supported according to [SWS_DM_00415] and the active session passes the execution permission check as per [SWS_DM_00101]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00417] Check supported DataIdentifier on active security level ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported on active security level if and only if the DataIdentifier is supported according to [SWS_DM_00415] and the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00418] Writing data for requested DataIdentifier ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), the Diagnostic Server shall write the data for a DataIdentifier according to its configuration as described in [SWS_DM_00505], [SWS_DM_00506], [SWS_DM_00407], [SWS_DM_00509]. ⌋*(SRS_Diag_04097)*

[SWS_DM_00419] Reaction on ApplicationError ⌈ If the external processor raised an ApplicationError contained in the ApplicationErrorSet UDSNegativeResponseCode, the Diagnostic Server shall return a negative response with the value of the errorCode of the raised ApplicationError. ⌋*(SRS_Diag_04196)*

7.2.1.6.11 Service 0x31 – RoutineControl

[SWS_DM_00201] Realization of UDS service RoutineControl (0x31) ⌈ The Diagnostic Server shall implement the diagnostic service RoutineControl (0x31) according to ISO 14229-1[1] for subFunctions startRoutine, stopRoutine and requestRoutineResults. ⌋*(SRS_Diag_04196)*

[SWS_DM_00202] Check for Supported RoutineIdentifier and Reaction ⌈ The Diagnostic Server shall check, whether the RoutineIdentifier addressed by the UDS Service RoutineControl (0x31) is supported by an existing DiagnosticRoutine with a matching id in the configuration. If the RoutineIdentifier addressed by the UDS Service RoutineControl (0x31) is not supported a negative response with NRC 0x31 (requestOutOfRange) shall be returned. ⌋*(SRS_Diag_04203)*

[SWS_DM_00448] Check supported RoutineIdentifier in active session ⌈ On reception of the UDS Service RoutineControl (0x31), a requested RoutineIdentifier shall be considered as supported in active session if and only if the RoutineIdentifier is supported according to [SWS_DM_00202] and the active session passes the execution permission check as per [SWS_DM_00101] and [SWS_DM_00102]. ⌋ *(SRS_Diag_04203)*

[SWS_DM_00437] Check supported RoutineIdentifier on active security level ⌈ On reception of the UDS Service RoutineControl (0x31), a requested DataIdentifier shall be considered as supported on active security level if and only if the RoutineIdentifier is supported according to [SWS_DM_00202] and the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00203] Check for Supported Subfunction and Reaction ⌈ The Diagnostic Server shall check, whether the Subfunction addressed by the UDS Service RoutineControl (0x31) is supported by checking the existence of the corresponding attributes start or stop or requestResult in the related DiagnosticRoutine configuration. If the Subfunction addressed by the UDS Service RoutineControl (0x31) is not supported by the configuration a negative response NRC 0x12 (SubfunctionNotSupported) shall be returned. ⌋*(SRS_Diag_04203)*

[SWS_DM_00210] UDS Service RoutineControl (0x31) startRoutine processing ⌈ The Diagnostic Server shall call the method Start of the interface RoutineService to process the subfunction startRoutine. ⌋*(SRS_Diag_04196)*

[SWS_DM_00211] UDS Service RoutineControl (0x31) requestRoutineResults processing ⌈ The Diagnostic Server shall call the method RequestResults of the interface RoutineService to process the subfunction requestRoutineResults. ⌋*(SRS_Diag_04196)*

[SWS_DM_00212] UDS Service RoutineControl (0x31) stopRoutine processing ⌈ The Diagnostic Server shall call the method Stop of the interface RoutineService to process the subfunction stopRoutine. ⌋*(SRS_Diag_04196)*

7.2.1.6.12 Service 0x34 – RequestDownload

[SWS_DM_00128] Realization of UDS service RequestDownload (0x34) ⌈ The Diagnostic Server shall implement the UDS service RequestDownload (0x34) according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196, SRS_Diag_04033)*

[SWS_DM_00446] Check Support of UDS service RequestDownload (0x34) in active session ⌈ On reception of the UDS service RequestDownload (0x34), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101]. ⌋ *(SRS_Diag_04203)*

[SWS_DM_00447] Check Support of UDS service RequestDownload (0x34) on active security level ⌈ On reception of the UDS service RequestDownload (0x34), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00131] UDS service RequestDownload (0x34) processing ⌈ The `DiagnosticServer` shall call the method `HandleMessage` of the interface `GenericUDSService` to process a UDS service RequestDownload (0x34). ⌋*(SRS_Diag_04196)*

7.2.1.6.13 Service 0x35 – RequestUpload

[SWS_DM_00134] Realization of UDS service RequestUpload (0x35) ⌈ The `DiagnosticServer` shall implement the UDS service RequestUpload (0x35) according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00438] Check Support of UDS service RequestUpload (0x35) in active session ⌈ On reception of the UDS service RequestUpload (0x35), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00439] Check Support of UDS service RequestUpload (0x35) on active security level ⌈ On reception of the UDS service RequestUpload (0x35), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋ *(SRS_Diag_04203)*

[SWS_DM_00136] UDS service RequestUpload (0x35) processing ⌈ The `DiagnosticServer` shall call the method `HandleMessage` of the interface `GenericUDSService` to process a UDS service RequestUpload (0x35). ⌋*(SRS_Diag_04033)*

7.2.1.6.14 Service 0x36 – TransferData

[SWS_DM_00137] Realization of UDS service TransferData (0x36) ⌈ The `DiagnosticServer` shall implement the UDS service TransferData (0x36) according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00440] Check Support of UDS service TransferData (0x36) in active session ⌈ On reception of the UDS service TransferData (0x36), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00441] Check Support of UDS service TransferData (0x36) on active security level ⌈ On reception of the UDS service TransferData (0x36), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋ *(SRS_Diag_04203)*

[SWS_DM_00138] UDS service TransferData (0x36) processing ⌈ The `Diagnostic Server` shall call the method `HandleMessage` of the interface `GenericUDSService` to process a UDS service TransferData (0x36). ⌋*(SRS_Diag_04033)*

ISO 14229-1[1] provides a UDS service TransferData (0x36) specific NRC evaluation sequence. This sequence has checks that in rotating order needs to be done by the `Diagnostic Server` and by the service processor itself. Therefore before actually running the service processor, the service processor needs means to do a certain verification step. As the `GenericUDSService` has only one single method this is not possible for the `GenericUDSService`. As a result of this, the entire service specific NRC handling is inside the `GenericUDSService` for UDS service TransferData (0x36).

[SWS_DM_00139] UDS service TransferData (0x36) validation ⌈ The `Diagnostic Server` shall realize all service specific `NRC` validation with the `GenericUDSService` of the service processors. ⌋*(SRS_Diag_04033)*

7.2.1.6.15 Service 0x37 – RequestTransferExit

[SWS_DM_00141] Realization of UDS service RequestTransferExit (0x37) ⌈ The `Diagnostic Server` shall implement the UDS service RequestTransferExit (0x37) according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00442] Check Support of UDS service RequestTransferExit (0x37) in active session ⌈ On reception of the UDS service RequestTransferExit (0x37), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101]. ⌋ *(SRS_Diag_04203)*

[SWS_DM_00443] Check Support of UDS service RequestTransferExit (0x37) on active security level ⌈ On reception of the UDS service RequestTransferExit (0x37), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00142] UDS service RequestTransferExit (0x37) processing ⌈ The `Diagnostic Server` shall call the method `HandleMessage` of the interface `GenericUDSService` to process a UDS service RequestTransferExit (0x37). ⌋ *(SRS_Diag_04033)*

[SWS_DM_00143] UDS service RequestTransferExit (0x37) validation ⌈ The Diagnostic Server shall realize all service specific NRC validation with the GenericUDSService of the service processors. ⌋*(SRS_Diag_04033)*

7.2.1.6.16 Service 0x3E – TesterPresent

[SWS_DM_00126] Realisation of UDS service 0x3E TesterPresent ⌈ The Diagnostic Server shall internally implement the diagnostic service 0x3E TesterPresent according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

7.2.1.6.17 Service 0x85 – ControlDTCSetting

The UDS service ControlDTCSetting is used by a client to stop or resume the updating of DTC status bits in the server.

[SWS_DM_00229] Support of UDS service ControlDTCSetting (0x85) ⌈ The Diagnostic Server shall provide the UDS service ControlDTCSetting (0x85) according to ISO 14229-1[1]. ⌋*(SRS_Diag_04180, SRS_Diag_04159)*

[SWS_DM_00444] Check Support of UDS service ControlDTCSetting (0x85) in active session ⌈ On reception of the UDS service ControlDTCSetting (0x85), a requested subfunction shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00445] Check Support of UDS service ControlDTCSetting (0x85) on active security level ⌈ On reception of the UDS service ControlDTCSetting (0x85), a requested subfunction shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00230] Check for supported subfunctions ⌈ If the Subfunction addressed by the UDS service ControlDTCSetting (0x85) according to [SWS_DM_00229] is not supported by the configuration, i.e., there is no DiagnosticControlDTCSetting configured with dtcSettingParameter matching the requested Subfunction value, the Diagnostic Server shall return a NRC 0x12 (SubfunctionNotSupported). ⌋*(SRS_Diag_04203)*

[SWS_DM_00231] Invalid value for optional request parameter ⌈ If the Diagnostic Server receives a UDS service ControlDTCSetting (0x85) request with DTCSettingControlOptionRecord != 0xFFFFFF, the Diagnostic Server shall send a NRC 0x31 (RequestOutOfRange). ⌋*(SRS_Diag_04203, SRS_Diag_04115)*

[SWS_DM_00232] Support of Subfunction 0x01 (ON) ⌈ If the Diagnostic Server receives a valid UDS service ControlDTCSetting (0x85) with Subfunction 0x01 (ON) and optionally with DTCSettingControlOptionRecord of value 0xFFFFFF, the Diagnostic Server shall:

- enable the update of the UDS DTC status byte
- enable the storage in event memory
- update the field ControlDTCStatus to kDTCSettingOn

⌋(*SRS_Diag_04180*, *SRS_Diag_04159*)

[SWS_DM_00233] Support of Subfunction 0x02 (OFF) ⌈ If the Diagnostic Server receives a valid UDS service ControlDTCSetting (0x85) with Subfunction 0x02 (OFF) and optionally with DTCSettingControlOptionRecord of value 0xFFFFFF, the Diagnostic Server shall:

- disable the update of the UDS DTC status byte
- disable the storage in event memory
- update the field ControlDTCStatus to kDTCSettingOff

, ⌋(*SRS_Diag_04180*, *SRS_Diag_04159*)

7.2.1.6.18 Service 0x86 – ResponseOnEvent

With the UDS Service ResponseOnEvent (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: UDS Service ReadDataByIdentifier 0x22).

Sub function ID	Sub-function name	Kind of sub-function	ServiceTo RespondTo	Support status
0x00/0x40	stopResponseOnEvent	Control		Supported
0x01/0x41	onDTCStatusChange	Setup	0x19, 0x0E	Supported
0x02/0x42	onTimerInterrupt	Setup		Not supported
0x03/0x43	onChangeOfDataIdentifier	Setup	0x22	Supported
0x04	reportActivatedEvents	Control		Supported
0x05/0x45	StartResponseOnEvent	Control		Supported
0x06/0x46	clearResponseOnEvent	Control		Supported
0x07/0x47	onComparisonOfValues	Setup	0x22	Supported
Other	OEM Specific	Setup		Not supported

Table 7.1: Supported sub function of ResponseonEvent (0x86)

[SWS_DM_00491] Realisation of UDS service 0x86 ResponseOnEvent ⌈ The DM shall internally implement the diagnostic service 0x86 ResponseOnEvent according to ISO 14229-1[1]. ⌋(*SRS_Diag_04160*)

[SWS_DM_00492] Client Server communication ⌈ The service ResponseOnEvent is related to a distinct client, i.e. the client performing the ResponseOnEvent initialisation receives the serviceToRespondTo-responses. ⌋(*SRS_Diag_04160*)

[SWS_DM_00493] Reestablishing of Client Server communication ⌈ In case of a canceled diagnostic conversation this client receives the serviceToRespondTo-responses after a successful reestablishing of a diagnostic conversation. ⌋ *(SRS_Diag_04160)*

[SWS_DM_00494] Supported sub functions of ResponseOnEvent service ⌈ The client can request different subfunctions of service ResponseOnEvent to initialised ResponseOnEvent services. The ECU supported subfunctions are listed in Table 7.1 Supported sub function of Response on Event (0x86). ⌋*(SRS_Diag_04160)*

[SWS_DM_00495] Start initialisation of ResponseOnEvent ⌈ The subfunction startResponseOnEvent shall always control all initialised ResponseOnEvent services. ⌋*(SRS_Diag_04160)*

[SWS_DM_00496] Stop initialisation of ResponseOnEvent ⌈ The subfunction stopResponseOnEvent shall always control all initialised ResponseOnEvent services. ⌋ *(SRS_Diag_04160)*

[SWS_DM_00497] Clear initialisation of ResponseOnEvent ⌈ The subfunction clearResponseOnEvent shall set the ResponseOnEvent services to status ResponseOnEvent-cleared. ⌋*(SRS_Diag_04160)*

[SWS_DM_00498] Exclusive ResponseOnEvent ressources ⌈ There is only one ResponseOnEvent resource per server which can be used by multiple clients. ⌋ *(SRS_Diag_04160)*

[SWS_DM_00499] Replacement of a not started ResponseOnEvent initialisation ⌈ If a new ResponseOnEvent initialisation is requested from a second client while a previous ResponseOnEvent initialisation is not started the new ResponseOnEvent initialisation replaces the previous ResponseOnEvent initialisation. ⌋*(SRS_Diag_04160)*

[SWS_DM_00500] Replacement of a started ResponseOnEvent initialisation ⌈ If a new ResponseOnEvent initialisation is requested from a second client while a previous ResponseOnEvent initialisation is active the ResponseOnEvent services have to be stopped and the new ResponseOnEvent initialisation replaces the previous ResponseOnEvent initialisation. ⌋*(SRS_Diag_04160)*

[SWS_DM_00501] Behavior while trying ResponseOnEvent activation while ResponseOnEvent is not initialised ⌈ A NRC 0x24 has to be sent if a ResponseOnEvent service is not initialised. ⌋*(SRS_Diag_04160)*

Note: The upcoming ISO 14229-1 will provides a more detailed description of ResponseOnEvent handling.

7.2.1.7 Cancellation of a Diagnostic Conversation

There are two root causes for the cancellation of a `Diagnostic Conversation`:

- Replacement by a newly requested Diagnostic Conversation according to [SWS_DM_00431],
- Cancellation externally triggered by call of `Cancel` method on the associated `DiagnosticConversation` Service Instance.
- Maximum number of busy responses reached (according to [SWS_DM_00369])

This section describes the actions to be performed on cancellation of a `Diagnostic Conversation`.

[SWS_DM_00482] Cancellation of a Diagnostic Conversation ⌈ Cancellation of a `Diagnostic Conversation` shall be performed according to [SWS_DM_00277], [SWS_DM_00278], [SWS_DM_00279], [SWS_DM_00280], [SWS_DM_00485]. ⌋ *(SRS_Diag_04167)*

[SWS_DM_00277] Cancellation of a Diagnostic Conversation in case of External Service Processing ⌈ On Cancellation of a `Diagnostic Conversation` in case a diagnostic request is currently processed on this `Diagnostic Conversation` by a service processor external to the `Diagnostic Server`, the `Diagnostic Server` shall notify this external service processor, that the processing for this service shall be canceled according to [SWS_DM_00042]. ⌋*(SRS_Diag_04167)*

[SWS_DM_00278] Cancellation of a Diagnostic Conversation in case of Internal Processing ⌈ On Cancellation of a `Diagnostic Conversation` in case a diagnostic request is currently processed on this protocol internally within the `Diagnostic Server`, the `Diagnostic Server` shall abort started activity as far as possible. ⌋ *(SRS_Diag_04167)*

[SWS_DM_00279] Cancellation of a Diagnostic Conversation before Response Transmission ⌈ On Cancellation of a `Diagnostic Conversation` in case a diagnostic request is currently processed on this protocol and response transmission has not yet been started, the `Diagnostic Server` shall skip sending any response, which implies **not** to call Transmit of the respective UDS Transport Protocol Handler. ⌋ *(SRS_Diag_04167)*

[SWS_DM_00280] Cancellation of a Diagnostic Conversation at Response Transmission ⌈ On Cancellation of a `Diagnostic Conversation` in case a diagnostic request is currently processed on this `Diagnostic Conversation` and the transmit functionality of the UDS TransportLayer was already called, nothing has to be done by the `Diagnostic Server`. This implies a sent out response. ⌋*(SRS_Diag_04167)*

[SWS_DM_00485] Reinitialization of Service Instance on Cancellation of a Diagnostic Conversation ⌈ On Cancellation of a `Diagnostic Conversation`, the `Diagnostic Server` shall reset the values of the fields of the associated `DiagnosticConversation` Service Instance according to [SWS_DM_00424]. ⌋ *(SRS_Diag_04167)*

[SWS_DM_00042] Canceling external service processors ⌈ External service processors, which are connected via the following service interfaces:

- GenericUDSService
- RoutineService
- SecurityAccess
- DataElement
- DataIdentifier
- VINInformation

shall be canceled by a call to the corresponding Cancel method. ⌋(*SRS_Diag_04167*)

7.2.2 Diagnostic Event Management

7.2.2.1 Diagnostic Events

7.2.2.1.1 Definition

Diagnostic events are used by applications to report the state of a monitored entity to the DM. An event uniquely identifies the monitored entity in the system. The DM receives event notifications from the applications and performs defined actions such as DTC status changes or capturing and storage of extended data records or snapshot records. In other words, events are the input source for the event memory management unit of the DM.

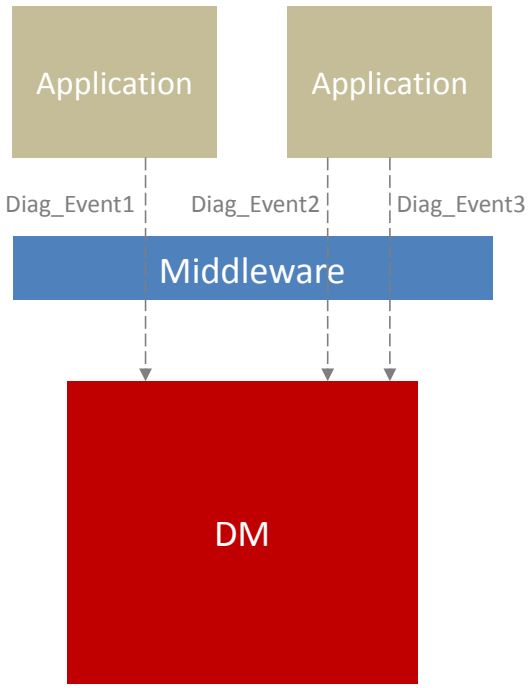

Figure 7.3: Example of diagnostic event usage

[SWS_DM_00007] Uniqueness of diagnostic events ⌈ An event is unique within the system and the DM shall only support notifications for a certain event from one single source. The implies that only one application can report a certain event and the event reporting interface is explicitly not re-entrant. ⌋*(SRS_Diag_04179)*

[SWS_DM_00008] Diagnostic event processing interface ⌈ The DM shall provide a service interface DiagnosticEvent per configured event. ⌋*(SRS_Diag_04179)*

The available events are derived from DiagnosticEvent.

[SWS_DM_00165] Considering only events referencing an DTC ⌈ The DM shall consider configured events according to [SWS_DM_00008] only if a DiagnosticEventToTroubleCodeUdsMapping exists referencing the event and a DiagnosticEventToTroubleCodeUdsMapping.troubleCodeUds. ⌋ *(SRS_Diag_04180)*

7.2.2.1.2 Monitors

A diagnostic monitor is a routine running inside an AA entity determining the proper functionality of a component. This monitoring function identifies a specific fault type (e.g. short-circuit to ground, missing signal, etc.) for a monitoring path. A monitoring path represents the physical system or a circuit, that is being monitored (e.g. sensor input). Each monitoring path is associated with exactly one diagnostic event.

In general diagnostic monitors are independent from the DM. Once the ECU is started and initialized they are permanently monitoring a part of the system and reporting the state to the DM. There are use cases where it might not be required to continue to monitor the system part and the monitor could stop it's task until a certain situation arises.

The DM uses the InitMonitor method of the DiagnosticMonitor service interface to trigger a (re-)initialization of diagnostic monitors in the AA.

[SWS_DM_00067] Monitor initialization for clearing reason ⌈ The DM shall call the InitMonitor method with the parameter reason set to kClear, in case an associated DTC, belonging to the current monitoring path, was actually cleared via the Clear method of the DiagnosticMemory service interface. ⌋*(SRS_Diag_04185)*

[SWS_DM_00068] Monitor initialization for operation cycle restart reason ⌈ When a monitored path was (re)started by the setting of the State field of the OperationCycle service interface to the value kStart, the DM shall call the InitMonitor method with the parameter reason set to value kRestart. ⌋*(SRS_Diag_04186)*

[SWS_DM_00069] Monitor initialization for enable condition re-enabling reason ⌈ In case an enable condition mapped to the diagnostic event was changed to fulfilled and this way all related enable conditions of the event were fulfilled, the DM shall call the InitMonitor method with reason parameter set to the value kReenabled. ⌋ *(SRS_Diag_04186)*

The detailed description of enable conditions can be found in section 7.2.2.4.3.

[SWS_DM_00070] Monitor initialization for DTC setting re-enabling reason ⌈ In case DTC setting is re-enabled via the UDS service request ControlDTCSetting - 0x85 (see ISO 14229-1[1]), the DM shall call the InitMonitor method with reason parameter set to the value kReenabled. ⌋*(SRS_Diag_04186)*

[SWS_DM_00071] Monitor initialization for storage condition reenabling reason ⌈ The DM shall shall call the InitMonitor method with reason parameter set to the value kStorageReenabled, if:

- a storage condition mapped to the diagnostic event was changed to fulfilled
- all related storage conditions of the event are fulfilled
- a kFailed or kPassed event was reported to the event while its storage conditions were disabled

⌋*(SRS_Diag_04186)*

The detailed description of storage conditions can be found in section 7.2.2.4.4.

7.2.2.1.3 Reporting

Diagnostic events are reported by applications via the ara::com event MonitorAction of service interface DiagnosticMonitor. The reported MonitorAction is processed by the DM, during the processing the diagnostic event and UDS DTC status bytes are calculated and DTC related data can be captured and stored in the fault memory. The DM provides also means to ignore a certain reported MonitorAction in some situations.

[SWS_DM_00168] Availability of DiagnosticMonitor service interfaces ⌈ The DM shall provide a service interface DiagnosticMonitor per configured DiagnosticEvent. ⌋*(SRS_Diag_04179)*

[SWS_DM_00167] Ignoring reported events for not started operation cycles ⌈ A new received MonitorAction ara::com event of service interface DiagnosticMonitor shall be discarded, if the referenced DiagnosticOperationCycle of this DiagnosticEvent (via DiagnosticEventToOperationCycleMapping) is set to END. ⌋*(SRS_Diag_04178)*

7.2.2.1.4 Debouncing

Debouncing of reported events is the capability of the DM to filter out undesirable noise reported by monitors. It can be configured on a per event basis.

There are two kind of different debounce algorithms implemented by the DM:

- Counter-based debouncing

- Time-based debouncing

[SWS_DM_00013] Events without debouncing ⌈ If an event is not referenced by any DiagnosticEventToDebounceAlgorithmMapping.diagnosticEvent, the DM shall not use a debounce algorithm for this event. ⌋*(SRS_Diag_04068)*

A monitoring application will report a MonitorActionType of kPrepassed or kPrefailed for events belonging to a common monitoring path, that are debounced by the DM.

[SWS_DM_00089] Reporting kPrepassed or kPrefailed for events without an assigned debouncing algorithm ⌈ A new received MonitorAction ara::com event with kPrepassed or kPrefailed for an diagnostic event without assigned debouncing algorithm, the DM shall interpret a reported kPrepassed as kPassed and kPrefailed as kFailed. ⌋*(SRS_Diag_04068)*

7.2.2.1.4.1 Counter-based debouncing

Counter-based debouncing is done on a per event based counting policy of reported PREPASSED or PREFAILED from diagnostic monitors. Per event an internal debounce counter is used. Passed or failed event states for events are calculated by evaluating configured thresholds of the internal debounce counter.

[SWS_DM_00014] Use of counter-based debouncing for events ⌈ A DiagnosticEvent shall be subject to counter-based debouncing if the DiagnosticEvent is referenced in the role diagnosticEvent by a DiagnosticEventToDebounceAlgorithmMapping, where the referenced debounceAlgorithm aggregates a DiagEventDebounceCounterBased in the role debounceAlgorithm. ⌋*(SRS_Diag_04068)*

[SWS_DM_00018] Internal debounce counter init and storage ⌈ If DiagnosticDebounceAlgorithmProps.debounceCounterStorage is set to false, the DM shall initialize the event's internal debounce counter to '0' upon startup. If DiagnosticDebounceAlgorithmProps.debounceCounterStorage is set to true, the DM shall initialize the event's internal debounce counter to the value stored in non-volatile memory. ⌋*(SRS_Diag_04124)*

[SWS_DM_00017] Calculation of the FDC based on the internal debounce counter ⌈ The DM shall calculate the FDC based on the value and range of the internal debounce counter by linear mapping. ⌋*(SRS_Diag_04125)*

[SWS_DM_00019] Internal debounce counter incrementation ⌈ The DM shall increment the event's internal debounce counter by the configured step-size value of DiagEventDebounceCounterBased.counterIncrementStepSize, when the monitor reports PREFAILED. ⌋*(SRS_Diag_04125)*

[SWS_DM_00024] Qualified failed event using counter-based debouncing ⌈ If the internal debounce counter is greater or equal to DiagEventDebounceCounterBased.counterFailedThreshold the DM shall process the event as FAILED. ⌋*(SRS_Diag_04125)*

[SWS_DM_00020] Internal debounce counter decrementation ⌈ The DM shall decrement the events internal debounce counter by the configured step-size value of DiagEventDebounceCounterBased.counterDecrementStepSize, when the related monitor reports kPrepassed as its MonitorAction ara::com event. ⌋ *(SRS_Diag_04125)*

[SWS_DM_00025] Qualified passed event using counter-based debouncing ⌈ If the internal debounce counter is less or equal to DiagEventDebounceCounterBased.counterPassedThreshold the DM shall process the event as PASSED. ⌋ *(SRS_Diag_04125)*

[SWS_DM_00021] Direct failed qualification of counter-based events ⌈ If the monitor reports FAILED, the DM shall set the internal debounce counter to the value DiagEventDebounceCounterBased.counterFailedThreshold. ⌋ *(SRS_Diag_04125)*

[SWS_DM_00029] Direct passed qualification of counter-based events ⌈ If the monitor reports PASSED, the DM shall set the internal debounce counter to the value DiagEventDebounceCounterBased.counterPassedThreshold. ⌋ *(SRS_Diag_04125)*

[SWS_DM_00022] Debounce counter jump up behavior ⌈ If DiagEventDebounceCounterBased.counterJumpUp is set to true for an event, the DM shall set the event's internal debounce counter to DiagEventDebounceCounterBased.counterJumpUpValue if PREFAILED is reported for this event and the current debounce counter value is less than DiagEventDebounceCounterBased.counterJumpUpValue. After setting the internal debounce counter to DiagEventDebounceCounterBased.counterJumpUpValue the processing according to [SWS_DM_00019] shall be done. ⌋*(SRS_Diag_04125)*

[SWS_DM_00023] Debounce counter jump down behavior ⌈ If PREPASSED is reported for this event and the current debounce counter value is greater than DiagEventDebounceCounterBased.counterJumpDownValue and counterJumpDown is set to true for an event, the DM shall set the event's internal debounce counter to DiagEventDebounceCounterBased.counterJumpDownValue. After setting the internal debounce counter to DiagEventDebounceCounterBased.counterJumpDownValue the processing according to [SWS_DM_00020] shall be done. ⌋ *(SRS_Diag_04125)*

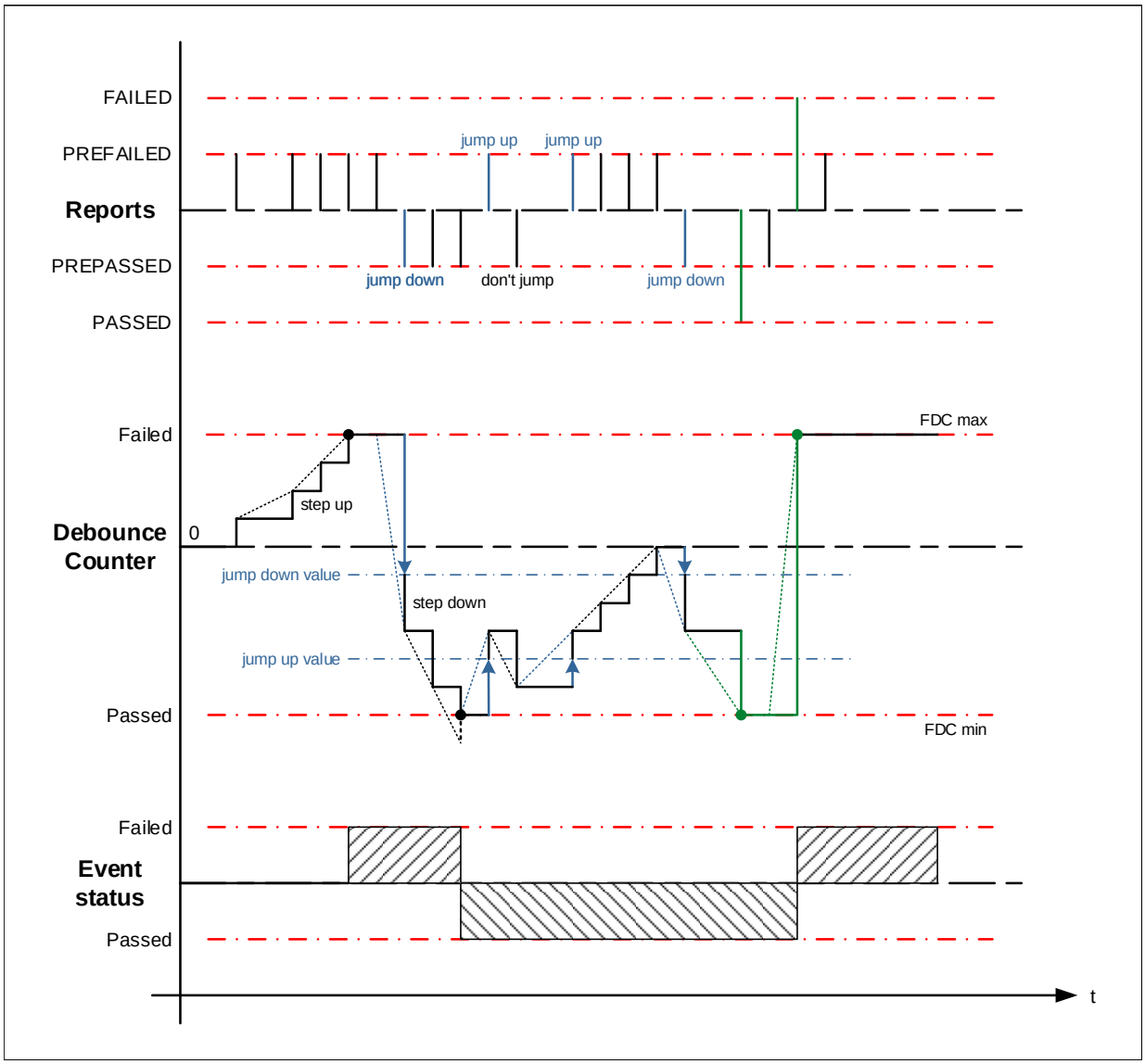

Figure 7.4: Counter-based debouncing

[SWS_DM_00028] Debounce counter persistency ⌈ If `DiagnosticDebounceAlgorithmProps.debounceCounterStorage` is set to True, the `DM` shall store the current value of the debounce counter in non-volatile memory. ⌋*(SRS_Diag_04124)*

7.2.2.1.4.2 Time-based debouncing

Time-based debouncing is done on a per event based counting policy of reported `PREPASSED` or `PREFAILED` from diagnostic monitors. Per event an internal debounce timer value is used. Passed or failed event states for events are calculated by evaluating configured thresholds of the internal debounce counter.

[SWS_DM_00015] Use of timer based debouncing for events ⌈ The existence of a `DiagnosticEventToDebounceAlgorithmMapping` with an aggregation of

DiagEventDebounceTimeBased by the referenced DiagnosticDebounceAlgorithmProps.debounceAlgorithm shall activate a time-based debouncing for this event. ⌋(*SRS_Diag_04068*)

[SWS_DM_00085] Internal debounce counter init ⌈ The DM shall initialize the event's internal debounce counter to '0' upon startup. ⌋(*SRS_Diag_04068*)

Note: DemDebounceCounterStorage is not supported for time-based debouncing.

[SWS_DM_00030] Calculation of the FDC based on the internal debounce timer ⌈ The DM shall calculate the FDC based on the value and range of the internal debounce timer by linear mapping. ⌋(*SRS_Diag_04068*)

The debounce counter is used to count upon a PREFAILED towards the qualified failed and upon a PREPASSED towards a qualified passed.

[SWS_DM_00031] Starting time-based event debouncing for failed ⌈ The DM module shall start the debounce timer when a PREFAILED was reported by a DiagnosticMonitor to qualify the reported event as FAILED only when the following conditions are met:

- The debounce timer for the event is not already counting towards FAILED.
- The event is not already qualified as FAILED.

⌋(*SRS_Diag_04068*)

[SWS_DM_00032] Restrictions on restarting a running event debounce timer for failed ⌈ If the debounce timer of a specific event is already counting towards FAILED, the DM shall not restart the debounce timer upon a further report of PREFAILED. ⌋ (*SRS_Diag_04068*)

[SWS_DM_00033] Debounce timer behavior upon reported failed ⌈ If the monitor reports FAILED, the DM shall set the debounce timer value to DiagEventDebounceTimeBased.timeFailedThreshold. ⌋(*SRS_Diag_04068*)

[SWS_DM_00034] Starting time-based event debouncing for passed ⌈ The DM module shall start the debounce timer when a PREPASSED was reported by a DiagnosticMonitor to qualify the reported event as PASSED only when the following conditions are met:

- The debounce timer for the event is not already counting towards PASSED.
- The event is not already qualified as PASSED.

⌋(*SRS_Diag_04068*)

[SWS_DM_00035] Restrictions on restarting a running event debounce timer for passed ⌈ If the debounce timer of a specific event is already counting towards PASSED, the DM shall not restart the debounce timer upon a further report of PREPASSED. ⌋ (*SRS_Diag_04068*)

[SWS_DM_00036] Debounce timer behavior upon reported passed ⌈ If the monitor reports PASSED, the DM shall set the debounce timer value to DiagEventDebounceTimeBased.timePassedThreshold. ⌋(*SRS_Diag_04068*)

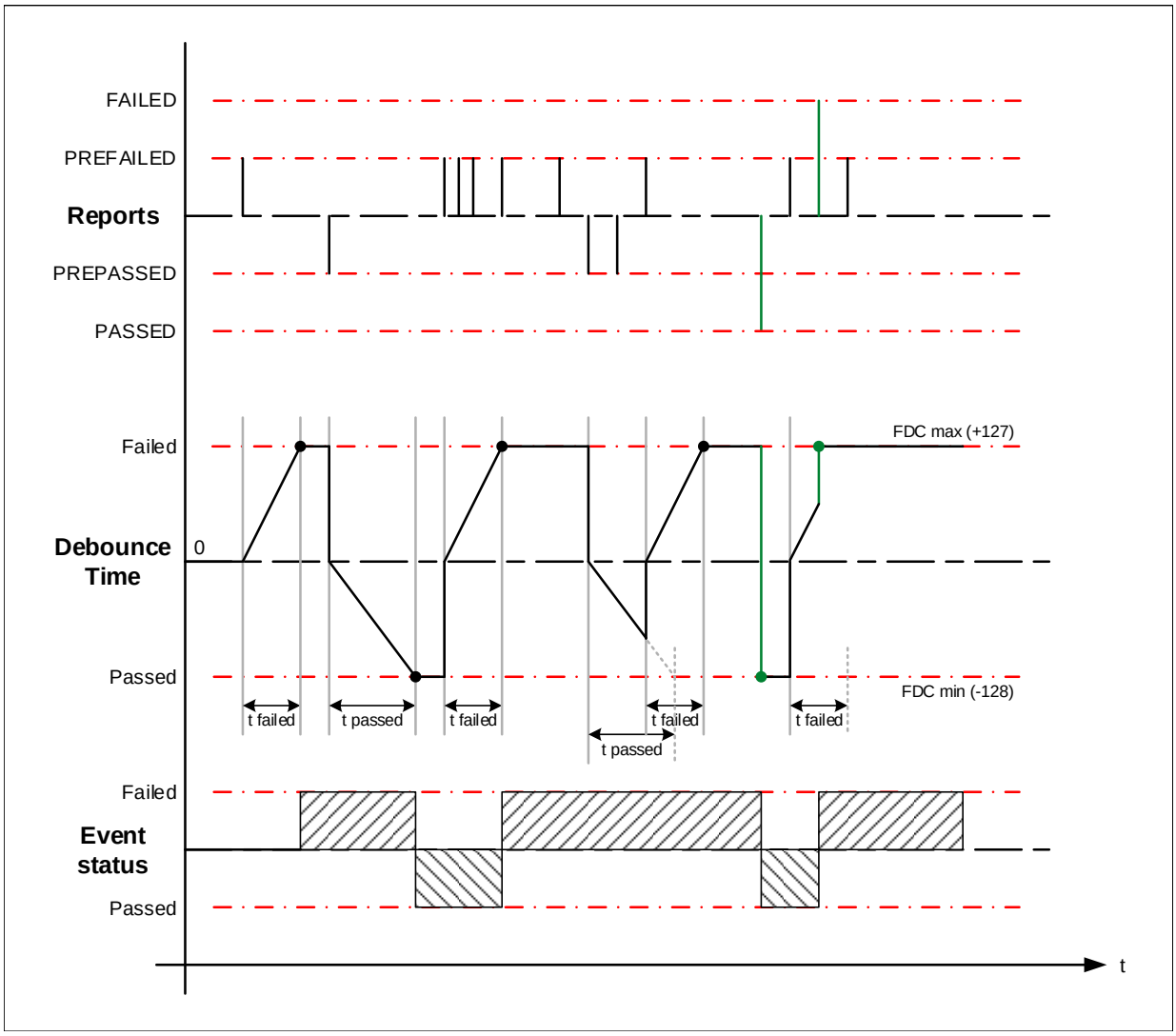

Figure 7.5: Timer based debouncing

[SWS_DM_00038] Continuing a frozen debounce timer ⌈ If a debounce timer is frozen and a new PREPASSED or PREFAILED is reported for this event, the DM module shall continue running the debounce timer starting with the frozen value. ⌋ (*SRS_Diag_04068*)

7.2.2.1.4.3 Debounce algorithm reset

In some situations the application might want to reset the debouncing or to freeze it. The DM provides the MonitorActions of service interface DiagnosticMonitor RESET_DEBOUNCING and FREEZE_DEBOUNCING to provide some means of external control of the debounce counter.

[SWS_DM_00040] Definition of debounce counter reset ⌈ To reset the debounce counter of an event, the DM shall set the corresponding debounce counter to zero. For time-based debouncing the debounce timer shall be stopped as well. ⌋ *(SRS_Diag_04068)*

[SWS_DM_00026] Application resetting the debounce counter ⌈ A new received `MonitorAction` ara::com event of service interface `DiagnosticMonitor` with RESET_DEBOUNCING shall reset the debounce counter. ⌋*(SRS_Diag_04068)*

While resetting a timer based debounce counter, it is regardless if the timer is counting towards a failed or passed.

[SWS_DM_00037] Debounce time freeze request ⌈ A new received `MonitorAction` ara::com event of service interface `DiagnosticMonitor` with FREEZE_DEBOUNCING shall freeze the related debounce timer for event with configured timer based debouncing. ⌋*(SRS_Diag_04068)*

[SWS_DM_00039] Resetting the debounce counter upon starting or restarting an operation cycle ⌈ If an operation cycle is started or restarted, the DM shall reset the debounce counter for all events referenced by `DiagnosticEventToOperationCycleMapping.diagnosticEvent` and referencing the started or restarted operation cycle by `DiagnosticEventToOperationCycleMapping.operationCycle`. ⌋*(SRS_Diag_04068)*

[SWS_DM_00086] Resetting the debounce counter after clearing DTC ⌈ If the DM executes a ClearDTC command, the DM shall reset the debounce counter for all events that have a `DiagnosticEventToTroubleCodeUdsMapping` to one of the cleared DTCs. ⌋*(SRS_Diag_04068)*

7.2.2.1.4.4 Dependencies to enable conditions

As described in section 7.2.2.4.3 enable conditions are used to suppress the result of reported event status information. Enable conditions have also effect on the debouncing behavior of the `DM`.

[SWS_DM_00087] Enable condition influence on debouncing behavior (freeze) ⌈ If an enable condition is not fulfilled for an event according to [SWS_DM_00074] and the debounce algorithm referenced by that event has the `DiagnosticDebounceAlgorithmProps.debounceBehavior` set to `freeze`, the DM shall freeze the according debounce counter or timer for the time the enabled condition is not fulfilled. This means that the debounce counter remains unchanged. ⌋*(SRS_Diag_04192)*

[SWS_DM_00377] Enable condition influence on debouncing behavior (reset) ⌈ If an enable condition is not fulfilled for an event according to [SWS_DM_00074] and the debounce algorithm referenced by that event has the `DiagnosticDebounceAlgorithmProps.debounceBehavior` set to `reset`, the DM shall reset the according debounce counter or timer and freeze it for the time the enabled condition is not fulfilled. ⌋*(SRS_Diag_04192)*

7.2.2.1.4.5 Dependencies to UDS service 0x85 ControlDTCSettings

[SWS_DM_00088] ControlDTCSetting influence (freeze) ⌈ If ControlDTCSetting is set to disabled for an event and the debounce algorithm referenced by that event has the `DiagnosticDebounceAlgorithmProps.debounceBehavior` set to `freeze`, the DM shall freeze the according debounce counter or timer for the time the ControlDTCSetting is set to disabled. This means that the debounce counter remains unchanged. ⌋*(SRS_Diag_04159)*

[SWS_DM_00378] ControlDTCSetting influence (reset) ⌈ If ControlDTCSetting is set to disabled for an event and the debounce algorithm referenced by that event has the `DiagnosticDebounceAlgorithmProps.debounceBehavior` set to `reset`, the DM shall reset the according debounce counter or timer and freeze it for the time the ControlDTCSetting is set to disabled. ⌋*(SRS_Diag_04159)*

7.2.2.2 DTC Status processing

7.2.2.2.1 Status processing

[SWS_DM_00213] DTC status processing ⌈

The `DM` shall process the `UDS` DTC status byte harmonizing with the ISO 14229-1[1] standard. ⌋*(SRS_Diag_04151)*

ISO 14229-1 Annex D generally defines status byte handling and the corresponding triggerings for them. The following requirements map interfaces and configuration parameters of the `DM` to generic `UDS` status bit transition descriptions.

[SWS_DM_00214] DTC status bit transitions triggered by test results ⌈ The `DM` shall process the `UDS` DTC status byte triggered by the test results reported via the `MonitorAction` event of the corresponding `DiagnosticMonitor` service interface. ⌋*(SRS_Diag_04151)*

Note that if configured, `kPrepassed` or `kPrefailed` status reports reported via `MonitorAction` trigger debounce mechanisms (see section 7.2.2.1.4). These status reports do not have direct impact on the `UDS` DTC status byte. If the status of an event gets fully qualified after debouncing (i.e.`kPassed` or `kFailed`), this information has the same impact on the status byte as it would have been reported via `MonitorAction`.

[SWS_DM_00215] Resetting the status of the `DTC` ⌈ The `DM` shall update the status of the `UDS` DTC status byte by setting **only** the `kTestFailed` bit to `FALSE` and leave all other bits unchanged, if the event `MonitorAction` is set to `kResetTestFailed`. ⌋*(SRS_Diag_04151)*

Rationale: This is an AUTOSAR-specific additional reset condition for the 'testFailed' bit.

[SWS_DM_00216] DTC status bit transitions triggered by operation cycle changes ⌈ The DM shall process the UDS DTC status byte triggered by operation cycle changes set in the OperationCycleState field of the corresponding OperationCycle service interface. ⌋*(SRS_Diag_04178)*

Note that Operation cycles are assigned to events by DiagnosticEventToOperationCycleMapping configuration items.

[SWS_DM_00217] DTC status bit transitions triggered by ClearDiagnosticInformation UDS service ⌈ The DM shall process the UDS DTC status byte triggered by the clearing of a DTC using the 0x14 ClearDiagnosticInformation UDS service. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

[SWS_DM_00218] Confirmation ⌈ The DM shall confirm the status of the UDS DTC status byte by setting the kConfirmedDTC bit to TRUE if the threshold determined by the corresponding DiagnosticEvent.eventFailureCycleCounterThreshold configuration parameter is reached. ⌋*(SRS_Diag_04180, SRS_Diag_04157, SRS_Diag_04067)*

Note that the TripCounter is processed according to the ISO 14229-1 specification.

If aging is supported for an event, the status is handled according to [SWS_DM_00243].

If there is an indicator mapped to the DTC, the 'warningIndicatorRequested' bit is handled as described in section 7.2.2.2.3.

7.2.2.2.2 Status change notifications

[SWS_DM_00219] Observability of the status byte ⌈ The DM shall provide the current status of events for the AA via the EventStatus field of the corresponding DiagnosticEvent service interface. ⌋*(SRS_Diag_04183)*

[SWS_DM_00220] Notification about DTC status changes ⌈ The DM shall update the AA for each DTC status changes by setting the EventStatus of the corresponding DiagnosticEvent service interface. ⌋*(SRS_Diag_04183)*

7.2.2.2.3 Indicators

[SWS_DM_00221] Handling indicator status ⌈ The DM shall handle the status of indicators assigned to events by the DiagnosticConnectedIndicator configuration item. ⌋*(SRS_Diag_04204)*

[SWS_DM_00222] Observability of indicator status ⌈ The DM shall provide the status of an indicator via the IndicatorStatus field of the corresponding Indicator service interface. ⌋*(SRS_Diag_04204)*

Note that the status of an indicator is determined by all the status information votes provided by events assigned to the corresponding indicator.

[SWS_DM_00223] Handling of 'warningIndicatorRequested' bit ⌈ The DM shall process the 'warningIndicatorRequested' bit of events and DTCs in accordance with the status vote for the assigned indicator. The 'warningIndicatorRequested' bit shall be set in case the status gets confirmed and consequently the events shall vote positively for setting the indicator. ⌋*(SRS_Diag_04204)*

For confirmation check [SWS_DM_00218].

[SWS_DM_00224] Indicator healing ⌈ The DM shall process indicator healing based on the DiagnosticIndicator.healingCycleCounterThreshold configuration parameter of the corrsponding indicator assigned to an event via DiagnosticConnectedIndicator.indicator. If the number of cycles (DiagnosticConnectedIndicator.healingCycle) in which the status was reported but not failed reaches the threshold, the 'warningIndicatorRequested' bit shall be set to 0, and the event shall vote negatively for the activation of the indicator. ⌋*(SRS_Diag_04204)*

7.2.2.2.4 User controlled WarningIndicatorRequest-bit

[SWS_DM_00476] User Controlled Warning IndicatorRequest-bit ⌈ The failsafe SW-C shall report the required WIR-status to DM (via the WIRStatus and has to ensure that the current WIR-status of an event (in DM) fits to the current failsafe-status in application:

- failsafe running: WIR-bit shall be set to "1"
- failsafe not running: WIR-bit shall be set to "0"

The failsafe SW-C has to report the status after every change of failsafe state. Each invocation of the WIRStatus in the Diagnostic Event Interface updated the WIR-bit for the corresponding events ⌋*()*

[SWS_DM_00477] Not Storing of 'warningIndicatorRequested' bit ⌈ Due to not storing the Status-Bit 7 ('warningIndicatorRequested' bit) on Shutdown, the failesafe Swc has to ensure that the 'warningIndicatorRequest' bit of an event fits to the current failsafe status after inizalization of the DM. ⌋*()*

7.2.2.3 Operation Cycles Management

The DM supports operation cycles according to ISO 14229-1[1]. Operation cycles have direct effect on the event memory behavior, such as calculation of event or DTC status.

Examples of typical operation cycles are:

- Ignition on/off cycles

- Power up/power down cycle
- Accumulated operating time cycles

Operation cycles are managed by the AA, the DM is notified about changes to operation cycle states using service interface OperationCycle.

[SWS_DM_00002] Automatic starting of operation cycles ⌈ If the configuration of DiagnosticOperationCycle.cycleAutostart is set to true, the DM shall set the respective State to kStart during the ECU startup and DM is initializing. ⌋ *(SRS_Diag_04178)*

A possible restart of the DM while the ECU is in operating mode, is not considered to trigger automatic restart of operation cycles.

[SWS_DM_00003] Automatic ending of operation cycles ⌈ If the configuration of DiagnosticOperationCycle.automaticEnd is set to true, the DM shall set the respective State to kStop while the ECU is shut down. ⌋*(SRS_Diag_04178)*

[SWS_DM_00004] Operation cycle persistency ⌈ If the configuration of DiagnosticOperationCycle.cycleStatusStorage is set to true, the DM shall persist the operation cycle State over the ECU power cycle. ⌋*(SRS_Diag_04178)*

[SWS_DM_00169] Restart of operation cycles ⌈ If the field State of the service interface OperationCycle is set to START and State was already set to START before, the DM shall restart the operation cycle and perform all steps triggered with a started operation cycle. ⌋*(SRS_Diag_04178)*

[SWS_DM_00192] Operation cycles are only ended once ⌈ If the field State of the service interface OperationCycle is set to END and State was already set to END before, the DM shall leave the State set to END and take no further actions. ⌋ *(SRS_Diag_04178)*

7.2.2.4 Event memory

The event memory is the database for faults detected by the system. It stores status information for events, DTCs and DTC related data. The DM uses the event memory for an ISO 14229-1[1] compliant handling of the fault memory.

There can be multiple event memories handled by the DM.

[SWS_DM_00055] Supported event memories ⌈ The DM shall support the

- primary event memory
- up to 256 user-defined event memories

according to ISO 14229-1[1]. ⌋*(SRS_Diag_04214)*

[SWS_DM_00056] Availability of the primary event memory ⌈ The DM shall support the primary event memory if a DTC exists having a DiagnosticMemoryDestinationPrimary referenced by its DiagnosticTroubleCodeProps.memoryDestination. ⌋*(SRS_Diag_04150)*

[SWS_DM_00057] Availability of a user-defined event memory ⌈ The DM shall support the user-defined event memory with the number DiagnosticMemoryDestinationUserDefined.memoryId if a DTC exists having a DiagnosticMemoryDestinationUserDefined with that user-defined number referenced by its DiagnosticTroubleCodeProps.memoryDestination. ⌋*(SRS_Diag_04214)*

7.2.2.4.1 DTC Introduction

A diagnostic trouble code (DTC) defines a unique identifier mapped to a diagnostic event. The DTC is used by diagnostics to uniquely identify data within the event memory database.

[SWS_DM_00060] Set of supported DTCs ⌈ The existence of a DiagnosticTroubleCodeUds indicates that the DM shall support this DTC. ⌋*(SRS_Diag_04201)*

Note: Due to DM restrictions the 'DiagnosticTroubleCodeObd' and 'DiagnosticTroubleCodeJ1939' are not supported.

7.2.2.4.1.1 Format

The DTC itself is a 3 byte value, that has different interpretations.

[SWS_DM_00058] DTC interpretation format ⌈ The DM shall use one internal DTC format interpretation that is defined in DiagnosticCommonProps.typeOfDtcSupported. ⌋*(TPS_DEXT_01008)*

[SWS_DM_CONSTR_00059] Restriction on supported DTC format ⌈ The DM shall support the following literals from interpreted DiagnosticCommonProps.typeOfDtcSupported (see also [SWS_DM_00058])

- iso11992_4
- iso14229_1
- saeJ2012_da

Further information about the format mapping is defined in [SWS_DM_00062].

The following literals are **not** supported by the DM:

- iso15031_6
- saeJ1939_73

⌋*(SRS_Diag_04201)*

7.2.2.4.1.2 Groups

Besides the term DTC, diagnostics uses DTC groups to address a range of single DTCs. A DTC group is defined by using a dedicated DTC value out of the range of valid DTCs to identify the group of DTCs.

A definition of valid DTC groups is provided by ISO 14229-1 [1] - Annex D.1. The DTC group is used in diagnostic just as any other DTC value, the DM internally resolved the DTC group and applies the requested operation to all DTCs of that group. The most common DTC group is the group of all DTCs, assigned to the DTC value 0xFFFFFF.

[SWS_DM_00064] Definition of DTC groups ⌈ The existence of a DiagnosticTroubleCodeGroup shall define the existence of the DTC group with the DTC identifier DiagnosticTroubleCodeGroup.groupNumber ⌋*(TPS_DEXT_03014)*

[SWS_DM_00065] Always supported availability of the group of all DTCs ⌈ The DM shall provide by default the DTC group 'GroupOfAllDTCs' assigned to the DTC group identifier 0xFFFFFF. This is DTC group contains always all configured DTCs. ⌋ *(SRS_Diag_04117)*

[SWS_DM_CONSTR_00082] Restriction on the configuration of the DTC group GroupOfAllDTCs ⌈ The Dm shall ignore any configuration of a DiagnosticTroubleCodeGroup.groupNumber with a value of 0xFFFFFF. ⌋*(SRS_Diag_04117)*

A configuration of the DTC group 0xFFFFFF via DiagnosticTroubleCodeGroup.groupNumber is not required. Within the DM basically all services and diagnostic requests having a DTC as input parameter accept also DTC group. As result of this, the operation is applied on all DTCs of that DTC group. To provide the reader a clear understanding if the DTC also can be a DTC group, it is explicitly mentioned in this specification. In case a DTC group is also valid, the DTC group definition of this chapter applies.

7.2.2.4.2 Destination

Each DTC is stored in one of the supported event memories according to [SWS_DM_00056] and [SWS_DM_00057].

[SWS_DM_00083] Event memory destination of an DTC ⌈ The existence of DiagnosticTroubleCodeProps.memoryDestination shall assign all DTCs referencing this DiagnosticTroubleCodeProps to the event memory referenced by DiagnosticTroubleCodeProps.memoryDestination. ⌋*(SRS_Diag_04150)*

[SWS_DM_CONSTR_00084] Each DTC shall be assigned to an event memory destination ⌈ The DM shall only support DTCs with a configured DiagnosticTroubleCodeProps.memoryDestination. ⌋*(SRS_Diag_04150)*

7.2.2.4.3 EnableConditions

DiagnosticEnableConditions are mapped to DiagnosticEvents by DiagnosticEventToEnableConditionGroupMappings.

[SWS_DM_00074] Handling of enable conditions ⌈ If any of the enable conditions mapped to the event are not fulfilled by a set State field, the DM shall ignore MonitorAction ara::com events. Otherwise (all of the enable conditions mapped to the event are fulfilled) shall accept and process the MonitorAction ara::com events. ⌋ *(SRS_Diag_04192)*

7.2.2.4.4 StorageConditions

In case the storage of event related data is triggered according to the DiagnosticCommonProps.memoryEntryStorageTrigger configuration parameter, or in case the update of event related data is triggered according to the DiagnosticFreezeFrame.trigger or DiagnosticExtendedDataRecord.trigger configuration parameters, the DM shall check all storage conditions assigned to the event.

DiagnosticStorageConditions are mapped to DiagnosticEvents by DiagnosticEventToStorageConditionGroupMappings.

[SWS_DM_00379] Handling of storage conditions ⌈ If any of the storage conditions mapped to the event are not fulfilled by a set State field, the DM shall ignore MonitorAction ara::com events. Otherwise (all of the storage conditions mapped to the event are fulfilled) shall accept and process the MonitorAction ara::com events. ⌋ *(SRS_Diag_04192)*

7.2.2.4.5 DTC related data

[SWS_DM_00148] Persistent storage of event memory entries ⌈ The DM shall be able to persistently store the status of all DTCs and its DTC related data:

- snapshot data if configured (at least one corresponding DiagnosticTroubleCodeProps.freezeFrame reference exists in the configuration)
- extended data if configured (at least one corresponding DiagnosticTroubleCodeProps.extendedDataRecord reference exists in the configuration)

⌋ *(SRS_Diag_04211)*

7.2.2.4.5.1 Triggering for data storage

[SWS_DM_00150] Primary trigger for event memory entry storage ⌈ Creating and storing memory entries (incl. collecting DTC-related data) shall be triggered according to the DiagnosticCommonProps.memoryEntryStorageTrigger configuration parameter. ⌋*(SRS_Diag_04211)*

Note that for updating snapshot record and extended data information record specific configuration options are available. For details check the following sections.

7.2.2.4.5.2 Storage of snapshot record data

[SWS_DM_00151] Snapshot record numeration ⌈ In case DiagnosticCommonProps.typeOfFreezeFrameRecordNumeration is set to calculated, snapshot records shall be numbered consecutively starting with 1 in their chronological order. If the parameter is set to configured, configured record numbers shall be used based on the DiagnosticFreezeFrame.recordNumber configuration parameters of the respective snapshot records. ⌋*(SRS_Diag_04205, SRS_Diag_04189)*

[SWS_DM_00152] Number of snapshot records for a DTC ⌈ In case DiagnosticCommonProps.typeOfFreezeFrameRecordNumeration is set to calculated, the number of snapshot records the DM is able to store for a DTC shall be determined by the DiagnosticTroubleCodeProps.maxNumberFreezeFrameRecords configuration parameter. In case DiagnosticCommonProps.typeOfFreezeFrameRecordNumeration is set to configured, the number of snapshot records is determined by the number of DiagnosticFreezeFrames configured for a DTC. ⌋*(SRS_Diag_04205, SRS_Diag_04190)*

Note that different snapshot records represent different snapshots collected in different points in time.

[SWS_DM_00153] Triggering for snapshot record storage ⌈ The data collection and the storage of the snapshot record shall be triggered by the DiagnosticFreezeFrame.trigger configuration parameter. The data layout of snapshot records is defined by the DiagnosticTroubleCodeProps.freezeFrameContent configuration class. Each referenced DiagnosticDataIdentifier shall be captured in its order via the DataIdentifier service interface. ⌋*(SRS_Diag_04205, SRS_Diag_04127)*

[SWS_DM_00273] Notification event upon `snapshot record` updates ⌈ After the DM has captured and stored a new snapshot record or overwritten an existing snapshot record with new data, the DM shall update the ara::com field SnapshotRecordUpdated of the service interface DTCInformation. ⌋*(SRS_Diag_04148)*

7.2.2.4.5.3 Storage of extended data

[SWS_DM_00154] Number of extended data for a DTC ⌈ The DM shall store zero or one extended data for a DTC. Extended data consists of extended data records. If at least one DiagnosticTroubleCodeProps.extendedDataRecord is configured for the corresponding DTC, the extended data shall be present in the event memory entry. ⌋*(SRS_Diag_04206, SRS_Diag_04190)*

Note that contrary to snapshot records, extended data records do not necessarily represent data collected in different points in time. Extended data consists of a configurable number of extended data records, which are all collected when the respective memory entry is created in the event memory. The update mechanism of extended data records is configurable.

[SWS_DM_00155] Extended data record numeration ⌈ Extended data record numbers shall always be determined by the configuration. The DiagnosticExtended-DataRecord.recordNumber configuration parameter defines the record number for each extended data record. ⌋*(SRS_Diag_04206, SRS_Diag_04189)*

[SWS_DM_00156] Triggering for extended data record storage and updates ⌈ The data collection and storage of the extended data record shall be triggered by the DiagnosticCommonProps.memoryEntryStorageTrigger trigger condition. Updating extended data records after being first stored, shall be configurable with the DiagnosticExtendedDataRecord.update configuration parameter. The data layout of extended data record is defined by the order of DiagnosticExtendedDataRecord.recordElement. Each DiagnosticDataElement shall be captured in its order via the DataElement service interface. ⌋*(SRS_Diag_04206, SRS_Diag_04127)*

7.2.2.4.6 Clearing DTCs

Clearing a DTC or a DTC group is the ability of the DM to reset the DTC status byte and deleting DTC assigned snapshot records and extended data records.

[SWS_DM_00116] Clearing a DTC group ⌈ When the DM is about to clear a DTC group is shall apply the same clear operation process as for a single DTC on all the DTCs of the DTC group which is cleared. ⌋*(SRS_Diag_04117)*

[SWS_DM_00117] Clearing a DTC ⌈ When the DM is about to clear a DTC it shall reset the event and UDS DTC status bytes and clear the snapshot records and extended data records stored for this DTC. ⌋*(SRS_Diag_04117)*

7.2.2.4.6.1 Locking of the DTC clearing process by an client

The DM supports more than one diagnostic clients as described in section 7.2.1.1.1. All configured clients can simultaneously send a ClearDTC diagnostic request. This chapter describes the DM behavior in this situations.

[SWS_DM_00144] Parallel clearing DTCs in different DiagnosticMemoryDestination ⌈ The DM shall support parallel clearing of DTCs if the target of the clear DTC operation is a different DiagnosticMemoryDestination. ⌋*(SRS_Diag_04117)*

[SWS_DM_00145] Allow only one simultaneous clear DTC operation for one DiagnosticMemoryDestination ⌈ If a diagnostic client is clearing the DTCs of a DiagnosticMemoryDestination the DM shall lock the clear DTC operation for all other clients requesting to clear the DTCs of the same DiagnosticMemoryDestination. ⌋*(SRS_Diag_04117)*

[SWS_DM_00146] Unlock clear DTC operation for one DiagnosticMemoryDestination ⌈ After the DM has finished the clear DTC operation, it shall unlock the clear DTC operation for this DiagnosticMemoryDestination. ⌋*(SRS_Diag_04117)*

[SWS_DM_00147] Behavior while trying to clear DTCs on a locked DiagnosticMemoryDestination ⌈ If the DM is requested to clear DTCs of a DiagnosticMemoryDestination and the DM has locked this DiagnosticMemoryDestination for clearing DTCs according to [SWS_DM_00144], the DM shall refuse the second clear DTC operation and shall return a NRC 0x22 (ConditionsNotCorrect). ⌋ *(SRS_Diag_04117)*

7.2.2.4.6.2 ClearConditions

In certain situations it is desirable to avoid that a DTC is cleared from the event memory. DiagnosticClearConditions are mapped to DTCs by DiagnosticTroubleCodeUdsToClearConditionGroupMappings.

[SWS_DM_00481] Handling of DiagnosticClearConditions ⌈ If any of the clear conditions mapped to the DTC to be cleared are not fulfilled by a set to false State field, the clear is forbidden. Otherwise (all of the clear conditions mapped to the DTC are fulfilled) the clear is allowed. ⌋*(SRS_Diag_04117)*

The effect of a forbidden clear DTC operation is described in the requirements below:

[SWS_DM_00123] Block status byte clearing during a clear DTC operation ⌈ If the DM is requested to clear a DTC with a forbidden clear according to [SWS_DM_00481] and a DiagnosticEventToTroubleCodeUdsMapping exists with a mapping from this DTC to an event and the event has DiagnosticEvent.clearEventBehavior set to noStatusByteChange, the DM shall not change the DTC status byte. ⌋ *(SRS_Diag_04117)*

[SWS_DM_00124] Limited status byte clearing during a clear DTC operation ⌈ If the DM is requested to clear a DTC with a forbidden clear according

to [SWS_DM_00481] and a DiagnosticEventToTroubleCodeUdsMapping exists with a mapping from this DTC to an event and the event has DiagnosticEvent.clearEventBehavior set to onlyThisCycleAndReadiness, the DM shall set the DTC status byte:

- Bit 1 TestFailedThisOperationCycle to '0'
- Bit 4 TestNotCompletedSinceLastClear to '1'
- Bit 5 TestFailedSinceLastClear to '0'
- Bit 6 TestNotCompletedThisOperationCycle to '1'.

⌋(*SRS_Diag_04117*)

[SWS_DM_00121] Forbidden clearing of snapshot records and extended data records ⌈ If the DM is requested to clear a DTC with a forbidden clear according to [SWS_DM_00481] the DM shall leave all snapshot records and extended data records for this DTC unchanged. ⌋(*SRS_Diag_04117*)

7.2.2.4.6.3 DTC clearing triggered by application

Besides the UDS request ClearDiagnosticInformation according to section 7.2.1.6.5.1 the DM supports the use case that the fault memory is cleared by an application call. One of the use cases is clearing of user-defined fault memory for diagnostic implementation without the ISO 14229-1[1] extension as described in section 7.2.1.6.5.1. This could be realized using a dedicated diagnostic routine service, whose application is in charge of the clearing process.

[SWS_DM_00260] instances of interface ClearDTC ⌈ The DM shall offer for every configured DiagnosticMemoryDestination a specific instance of the DiagnosticMemory interface. ⌋(*SRS_Diag_04194*)

[SWS_DM_00262] Common semantic behavior for ClearDTC triggered via diagnostics or application ⌈ The clear DTC operation itself is semantically identical, independent if triggered via diagnostic service or application method call. All requirements for clear DTC apply in either case. ⌋(*SRS_Diag_04194*)

[SWS_DM_00261] Usage of ClearDTC Interface ⌈ If the method Clear of the DiagnosticMemory interface is called, the DM shall clear the DTC or DTC group provided in the parameter DTC. The clear DTC shall clear the fault memory associated to the instance of the clearDTC interface only. ⌋(*SRS_Diag_04194*)

[SWS_DM_00263] ClearDTC call on invalid DTC or DTCgroup ⌈ If the method Clear of the DiagnosticMemory interface is called and the parameter DTC has no matching configured DTC group according to [SWS_DM_00064] or configured DTC by DiagnosticTroubleCodeUds.udsDtcValue, the DM shall trigger the error WRONG_DTC for that method call. ⌋(*SRS_Diag_04194*)

[SWS_DM_00265] ClearDTC called while another clear operation is in progress ⌈ If the method `Clear` of the `DiagnosticMemory` interface is called and another clear DTC operation is currently in progress, the DM shall trigger the error BUSY. ⌋ *(SRS_Diag_04194)*

[SWS_DM_00266] ClearDTC processing in case of memory errors ⌈ If the method `Clear` of the `DiagnosticMemory` interface is called and the DM detects physical memory errors and thus cannot guarantee that the clear operation was done successfully, the DM shall trigger the error MEMORY_ERROR. ⌋*(SRS_Diag_04194)*

[SWS_DM_00267] Possible failure of ClearDTC ⌈ If the method `Clear` of the `DiagnosticMemory` interface is called and the clear operation fails due to the reasons according to [SWS_DM_00122], the DM shall trigger the error FAILED. ⌋ *(SRS_Diag_04194)*

7.2.2.4.7 Aging

[SWS_DM_00237] Aging ⌈ The `DM` shall only support `aging` for DTCs if the corresponding `DiagnosticTroubleCodeProps.agingAllowed` configuration parameter is set. ⌋*(SRS_Diag_04133)*

[SWS_DM_00238] Aging and healing ⌈ If an indicator is configured for the corresponding event, the process of aging (counting of aging counter) shall be started only after the healing is completed ('warningIndicatorRequested' bit is set to 0). ⌋ *(SRS_Diag_04133)*

[SWS_DM_00239] Aging counter ⌈ The `DM` shall support an aging counter for each event memory entry. ⌋*(SRS_Diag_04133)*

Note that this counter shall be available as internal data element of extended data or snapshot record.

[SWS_DM_00240] Processing the aging counter ⌈ The `DM` shall only allow processing the aging counter if the related DTC is stored in the event memory and the status is qualified as passed ('testFailed' bit is set to 0). ⌋*(SRS_Diag_04133)*

[SWS_DM_00241] Aging cycle and threshold ⌈ The `aging` shall be calculated based on the referred DiagnosticOperationCycle via the reference `DiagnosticAging.agingCycle`. The `DiagnosticAging.threshold` defines the number of aging cycles until aging. If `agingRequiresTestedCycle` is set the cycle shall only be considered in which the status was reported but not failed ('testNotCompletedThisOperationCycle' bit and 'testFailedThisOperationCycle' bit are set to 0). If the threshold is reached, the event memory entry shall be deleted (aged) from the event memory. ⌋ *(SRS_Diag_04133)*

[SWS_DM_00242] Re-occurrence after aging ⌈ The `DM` shall handle the re-occurrence of unlearned events like new events, since they were previously deleted from the event memory by aging. ⌋*(SRS_Diag_04133)*

[SWS_DM_00243] Aging-related UDS DTC status byte processing ⌈ As a consequence of aging, the DM shall set the following status bits to 0:

- 'confirmedDTC' unconditionally
- 'testFailedSinceLastClear' conditionally, if statusBitHandlingTestFailedSinceLastClear is set to statusBitAgingAndDisplacement

⌋(*SRS_Diag_04140*)

7.2.2.4.8 NumberOfStoredEntries

[SWS_DM_00651] NumberOfStoredEntries ⌈ The DM shall provide the field NumberOfStoredEntries of DiagnosticEventMemory to return the number of event memory entries (DTCs) currently stored in the primary event memory where the status of a DTC is pendingDTC = 1 and/or confirmedDTC = 1. An update notification shall be sent whenever the number of NumberOfStoredEntries has changed. ⌋(*SRS_Diag_04109*)

Note: The reported number of NumberOfStoredEntries shall be identical to the response of ReadDTCInformation (0x19) service with sub-function 0x01 (reportNumberOfDTCByStatusMask) and a DTCStatusMask set to 0x0C.

7.2.3 Required Configuration

The Autosar Diagnostic Extract Template (DEXT) [2] is used for the DM configuration. By design this format is made as exchange format between the tools in the diagnostic workflow, in different steps data is added. To accommodate the fact that data is incomplete and refined in a later step, the DEXT [2] allows most of the elements to be optional and added at a later point in time. However at the point in time, when the DEXT [2] is used to configure the DM, a certain minimum content is required. In this chapter a loose list of DEXT [2] constraints is given. The mentioned elements need to be present so that the DM can be configured. Also the reaction on such missing elements is implementation specific, it is stated that the DM will not be able to behave as described in the document. A possible but not mandatory reaction is to refuse the DM generation at all and forcing the user to provide complete data.

[SWS_DM_CONSTR_00168] Required operation cycles for diagnostic events ⌈ Each DiagnosticEvent requires exactly one DiagnosticEventToOperationCycleMapping referencing the diagnosticEvent and one DiagnosticOperationCycle. ⌋(*SRS_Diag_04178*)

[SWS_DM_CONSTR_00206] Supported format for data identifier for VINDataIdentifier ⌈ A DiagnosticDataIdentifier with representsVin set to true, requires that it aggregates only one DiagnosticParameter which itself aggregates a DiagnosticDataElement having a 17 byte uint8 array as baseType. ⌋ (*SRS_Eth_00026*)

7.2.4 Diagnostic Data Management

In various situations, the Diagnostic Server facilitates reading or writing of particular diagnostic data. One needs to distinguish between internal and external diagnostic data. By definition, internal data is managed by the Diagnostic Server itself, and external data is managed by external applications. In the latter case, communication between Diagnostic Server and the external application takes place via Service Interfaces. There are several Service Interfaces defined concerning diagnostic data.

The purpose of this chapter is to describe the supported use-cases for handling diagnostic data and the way how to configure each use-case within the DEXT.

Recall that a DiagnosticDataIdentifier is composed of DiagnosticParameters each of which aggregates a single DiagnosticDataElement. In different use cases, it is required to manage diagnostic data either on the level of DiagnosticDataIdentifier or on the fine granular level of DiagnosticDataElements.

7.2.4.1 Internal and External Diagnostic Data Elements

A DiagnosticDataElement is called internal if there exists a DiagnosticDemProvidedDataMapping referencing this DiagnosticDataElement, otherwise it is called an external DiagnosticDataElement.

Table 7.2 gives a list of the supported internal DiagnosticDataElements, where

Data Provider refers to the NameToken defined in the role of dataProvider of the associated DiagnosticDemProvidedDataMapping,

Content describes the actual content of the data,

Format describes the data format of the DiagnosticDataElement.

Context defines the exclusive context in which this DiagnosticDataElement is defined (if applicable).

Data Provider	Content	Format	Context
DEM_AGINGCTR_DOWNCNT	Down-counting aging counter of contextual DTC	1 byte	DEM
DEM_AGINGCTR_UPCNT	Up-counting aging counter of contextual DTC	1 byte	DEM
DEM_AGINGCTR_UPCNT_FIRST_ACTIVE	Up-counting aging counter of contextual DTC, starting at 1 when aging starts	1 byte	DEM
DEM_CURRENT_FDC	Fault Detection Counter of contextual DTC	1 byte	DEM
DEM_CYCLES_SINCE_FIRST_FAILED	Operation Cycle Counter of contextual DTC– Cycles since first failed	1 byte	DEM
DEM_CYCLES_SINCE_LAST_FAILED	Operation Cycle Counter of contextual DTC – Cycles since last failed	1 byte	DEM
DEM_FAILED_CYCLES	Operation Cycle Counter of contextual DTC – Failed cycles	1 byte	DEM

DEM_MAX_FDC_DURING_CURRENT_CYCLE	Fault Detection Counter maximum value during current operation cycle of contextual DTC	1 byte	DEM
DEM_MAX_FDC_SINCE_LAST_CLEAR	Fault Detection Counter maximum value since last clear of contextual DTC	1 byte	DEM
DEM_OCCCTR	Occurrence counter of contextual DTC	1 byte	DEM
DEM_OVFLIND	Overflow indication of contextual DTC (0 = False, 1 = True)	1 byte	DEM
DEM_SIGNIFICANCE	Event significance of contextual DTC (refer to DemDTCSignificance) (0 = OCCURRENCE, 1 = FAULT)	1 byte	DEM
DEM_PRIORITY	Priority of the contextual DTC	1 byte	DEM
DCM_SESSION	Current session of contextual DiagnosticConversation	1 byte	DCM
DCM_SECURITY_LEVEL	Current security level of contextual DiagnosticConversation	1 byte	DCM

Table 7.2: Supported internal DiagnosticDataElements

[SWS_DM_00393] Retrieving data for internal DiagnosticDataElements ⌈ If DM requires to provide or store data configured as internal DiagnosticDataElement which is supported by the Diagnostic Server according to Table 7.2, then DM shall use the respective internally managed data value as defined in Table 7.2. ⌋ *(SRS_Diag_04097)*

[SWS_DM_CONSTR_00394] Internal DiagnosticDataElements are read-only ⌈ A DiagnosticDataIdentifier referenced by a DiagnosticWriteDataByIdentifier service shall not contain any internal DiagnosticDataElement. ⌋*(SRS_Diag_04097)*

An internal DiagnosticDataElement is called DCM-exclusive resp. DEM-exclusive if the context of the name token described in Table 7.2 is set accordingly. The implicit restriction of such DiagnosticDataElements to the context in which they are defined is made explicit in the following requirements. These requirements are formulated in a way that Table 7.2 might in future be extended by internal DiagnosticDataElements not restricted to exclusive use within a DCM resp. DEM context.

[SWS_DM_CONSTR_00395] Restriction on DEM-exclusive DiagnosticDataElements ⌈ A DiagnosticParameter containing a DEM-exclusive internal DiagnosticDataElement shall not be contained in a DiagnosticDataIdentifier referenced by a DiagnosticReadDataByIdentifier, nor shall it be contained in a realization of DiagnosticRoutineSubfunction. ⌋*(SRS_Diag_04097)*

[SWS_DM_CONSTR_00396] Restriction on DCM-exclusive DiagnosticDataElements ⌈ A DiagnosticParameter containing a DCM-exclusive internal DiagnosticDataElement shall not be contained in a DiagnosticDataIdentifier referenced by a DiagnosticDataIdentifierSet which is referenced by some DiagnosticTroubleCodeProps in the role of freezeFrameContent, nor shall it be contained in a DiagnosticExtendedDataRecord. ⌋*(SRS_Diag_04097)*

Note: The notion of internal and external is exclusively defined for DiagnosticDataElements and does not apply to DiagnosticDataIdentifier.

[SWS_DM_00397] Retrieving data for external DiagnosticDataElements ⌈ If the Diagnostic Server is required to read data configured as external DiagnosticDataElement, then the Diagnostic Server shall utilize the associated RPortPrototype typed by the DataElement Service Interface and call its Read method. ⌋*(SRS_Diag_04097)*

Note: In general, there are multiple instances of DataElement Service Interface available in the running system. Which instance to choose for the given request to read an external DiagnosticDataElement is part of system integration. Support for this integration is provided by DiagnosticMappings described in section 7.2.4.2.1.

7.2.4.2 Reading and Writing Diagnostic Data Identifier

The Diagnostic Server supports multiple ways to read or write diagnostic data defined as DiagnosticDataIdentifier:

- reading each DiagnosticDataElement contained in the DiagnosticDataIdentifier independently as described in section 7.2.4.1,
- reading or writing the DiagnosticDataIdentifier as a whole via the DataIdentifier service interface,
- reading or writing the DiagnosticDataIdentifier as a whole via the GenericUDSService service interface.

The method to choose between these ways of data handling is by configuration of DiagnosticMappings refering to the DiagnosticDataIdentifier. This chapter describes the supported DiagnosticMappings and provides requirements on reading and writing DiagnosticDataIdentifier reflecting the short description above.

7.2.4.2.1 Supported Diagnostic Mappings

There are multiple types of DiagnosticMappings related to DiagnosticDataElements and DiagnosticDataIdentifier:

DiagnosticMapping	diagnostics endpoint	target endpoint
DiagnosticDemProvidedDataMapping	DiagnosticDataElement	DM internal data provider
DiagnosticServiceDataMapping	DiagnosticDataElement	DataPrototype
DiagnosticServiceSwMapping	DiagnosticDataElement	SwcServiceDependency
DiagnosticServiceDataIdentifierMapping	DiagnosticDataIdentifier	SwcServiceDependency

Table 7.3: Diagnostic Mappings

The DiagnosticDemProvidedDataMapping is used to distinguish between internal and external DiagnosticDataElement as described in section 7.2.4.1.

The DiagnosticServiceDataMapping is currently not supported as input for the configuration of the Diagnostic Server.

The DiagnosticServiceSwMapping maps a DiagnosticDataElement in the role of diagnosticDataElement to a SwcServiceDependency in the role of mappedSwcServiceDependencyInExecutable.

Note: The DiagnosticServiceSwMapping also provides an indirect reference to a DiagnosticDataIdentifier by means of referencing a DiagnosticDataByIdentifier in the role of serviceInstance which itself references the DiagnosticDataIdentifier in the role of dataIdentifier. However, this variant of configuration shall not be used on AP, instead DiagnosticServiceDataIdentifierMapping shall be used. Main rational for this restriction is that DiagnosticServiceSwMapping would allow for different configurations for reading and writing the same DiagnosticDataIdentifier. Instead, the DiagnosticServiceDataIdentifierMapping shall be used to map a DiagnosticDataIdentifier to some application port.

The DiagnosticServiceDataIdentifierMapping maps a DiagnosticDataByIdentifier in the role of diagnosticDataIdentifier to a SwcServiceDependency in the role of mappedSwcServiceDependency.

Details regarding the modeling of diagnostic mappings can be found in the TPS Manifest Specification [12].

7.2.4.2.2 Reading Diagnostic Data Identifier

[SWS_DM_00401] Reading Diagnostic Data Identifier on Data Element level ⌈ If the Diagnostic Server is required to read data configured as DiagnosticDataIdentifier and at least on of the DiagnosticDataElements aggregated in this DiagnosticDataIdentifier is referenced by some DiagnosticMapping, then Diagnostic Server shall retrieve the data by reading data from each DiagnosticDataElement separately according to [SWS_DM_00393] and [SWS_DM_00397]. ⌋*(SRS_Diag_04097)*

[SWS_DM_00503] Reading Diagnostic Data Identifier by DataIdentifier interface ⌈ If the Diagnostic Server is required to read data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticServiceDataIdentifierMapping of category DATA_IDENTIFIER, then the Diagnostic Server shall use the DataIdentifier interface associated to the DiagnosticDataIdentifier for reading the data. ⌋*(SRS_Diag_04097)*

[SWS_DM_00504] Reading Diagnostic Data Identifier by GenericUDSService interface ⌈ If the Diagnostic Server is required to read data configured as

DiagnosticDataIdentifier which is referenced by a DiagnosticService-DataIdentifierMapping of category GENERIC_UDS_SERVICE, then the Diagnostic Server shall use the instance of the GenericUDSService interface referenced by the DiagnosticServiceDataIdentifierMapping and call its HandleMessage method with SID set to 0x22 and requestData set to the id of the DiagnosticDataIdentifier. The responseData is respectively composed of the requested id and the content of every dataRecord of the related dataElement. ⌋(*SRS_Diag_04097*)

[SWS_DM_00404] Default Service Interface for reading DiagnosticDataIdentifier ⌈ If the Diagnostic Server is required to read data configured as DiagnosticDataIdentifier and none of the requirements [SWS_DM_00401], [SWS_DM_00503], [SWS_DM_00504] applies, then the Diagnostic Server shall utilize the associated RPortPrototype typed by the DataIdentifier Service Interface and call its Read method. ⌋(*SRS_Diag_04097*)

Note: The default configuration as described in [SWS_DM_00404] assumes that there is a single instance of PPortPrototype defined in the system matching the RPort-Prototype associated to the requested DiagnosticDataIdentifier as defined in section 9.3.4.2. In this case, it is part of integration step to link these two ports.

7.2.4.2.3 Writing Diagnostic Data Identifier

[SWS_DM_00505] Writing Diagnostic Data Identifier by DataIdentifier interface ⌈ If the Diagnostic Server is required to write data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticServiceDataIdentifierMapping of category DATA_IDENTIFIER, then the Diagnostic Server shall use the DataIdentifier interface associated to the DiagnosticDataIdentifier for writing the data. ⌋(*SRS_Diag_04097*)

[SWS_DM_00506] Writing Diagnostic Data Identifier by GenericUDSService interface ⌈ If the Diagnostic Server is required to writing data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticService-DataIdentifierMapping of category GENERIC_UDS_SERVICE, then the Diagnostic Server shall use the instance of the GenericUDSService interface referenced by the DiagnosticServiceDataIdentifierMapping and call its HandleMessage method with SID set to 0x2E and requestData set to the id of this DiagnosticDataIdentifier followed by the data to be written to this DiagnosticDataIdentifier. ⌋(*SRS_Diag_04097*)

[SWS_DM_00407] Default Service Interface for writing DiagnosticDataIdentifier ⌈ If the Diagnostic Server is required to write data configured as DiagnosticDataIdentifier and none of the requirements [SWS_DM_00505], [SWS_DM_00506] applies, then the Diagnostic Server shall utilize the associated RPortPrototype typed by the DataIdentifier Service Interface and call its Write method. ⌋(*SRS_Diag_04097*)

Note: The default configuration as described in [SWS_DM_00407] assumes that there is a single instance of PPortPrototype defined in the system matching the RPortPrototype associated to the requested DiagnosticDataIdentifier as defined in section 9.3.4.2. In this case, it is part of integration step to link these two ports.

7.2.4.2.4 Reading and writing VIN data

[SWS_DM_00508] Reading DiagnosticDataIdentifier configured for representing VIN ⌈ If the Diagnostic Server needs to read data configured as DiagnosticDataIdentifier with attribute representsVin set to true, the Diagnostic Server shall obtain it by using the method Read of the service interface VINInformation. ⌋*(SRS_Diag_04097)*

[SWS_DM_00509] Writing DiagnosticDataIdentifier configured for representing VIN ⌈ If the Diagnostic Server needs to write data configured as DiagnosticDataIdentifier with attribute representsVin set to true, the Diagnostic Server shall call the method Write of the service interface VINInformation. ⌋ *(SRS_Diag_04097)*

8 API specification

8.1 C++ UDS Transportlayer API Interfaces

This chapter lists all provided and required C++ API interfaces of the DM for interaction with a UDS Transportlayer implementation.

8.1.1 UDS Transportlayer Types

8.1.1.1 uds_transport::ByteVector

[SWS_DM_00338] ⌈ The type alias ara::diag::uds_transport::ByteVector is defined in Table 8.1. ⌋()

Kind:	type alias
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef ara::core::Span<uint8_t>
Syntax:	using ara::diag::uds_transport::ByteVector = ara::core::Span<uint8_t>;
Header file:	#include "ara/diag/uds_transport/protocol_types.h"
Description:	This is the type of ByteVector.

Table 8.1: type alias ara::diag::uds_transport::ByteVector

8.1.1.2 uds_transport::ChannelID

[SWS_DM_00337] ⌈ The type alias ara::diag::uds_transport::ChannelID is defined in Table 8.2. ⌋()

Kind:	type alias
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef uint32_t
Syntax:	using ara::diag::uds_transport::ChannelID = uint32_t;
Header file:	#include "ara/diag/uds_transport/protocol_types.h"
Description:	

Table 8.2: type alias ara::diag::uds_transport::ChannelID

8.1.1.3 uds_transport::Priority

[SWS_DM_00451] ⌈ The type alias ara::diag::uds_transport::Priority is defined in Table 8.3. ⌋()

Kind:	type alias
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef uint8_t
Syntax:	using ara::diag::uds_transport::Priority = uint8_t;
Header file:	#include "ara/diag/uds_transport/protocol_types.h"
Description:	

Table 8.3: type alias ara::diag::uds_transport::Priority

8.1.1.4 uds_transport::ProtocolKind

[SWS_DM_00452] ⌈ The type alias ara::diag::uds_transport::ProtocolKind is defined in Table 8.4. ⌋*()*

Kind:	type alias
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef ara::core::String
Syntax:	using ara::diag::uds_transport::ProtocolKind = ara::core::String;
Header file:	#include "ara/diag/uds_transport/protocol_types.h"
Description:	

Table 8.4: type alias ara::diag::uds_transport::ProtocolKind

8.1.1.5 uds_transport::UdsMessageConstPtr

[SWS_DM_00304] ⌈ The type alias ara::diag::uds_transport::UdsMessageConstPtr is defined in Table 8.5. ⌋*()*

Kind:	type alias
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef std::unique_ptr<const UdsMessage, std::function<void(const UdsMessage*)»
Syntax:	using ara::diag::uds_transport::UdsMessageConstPtr = std::unique_ptr<const UdsMessage, std::function<void(const UdsMessage*)»;
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	This is the unique_ptr for constant UdsMessages containing a custom deleter as provided by the generic/core DM part towards the UdsTransportLayer-Plugin. How the exact typedef for UdsMessageConstPtr looks like, is up to the DM product vendor. I.e. how f.i. the deleter signature looks like ... basically the minimal agreement is: UdsMessage ConstPtr shall behave like a std::unique_ptr<const UdsMessage>!
Notes:	How the exact typedef for UdsMessageConstPtr looks like, is up to the DM product vendor. I.e. how f.i. the deleter signature looks like ... basically the minimal agreement is: UdsMessage ConstPtr shall behave like a std::unique_ptr<const UdsMessage>!

Table 8.5: type alias ara::diag::uds_transport::UdsMessageConstPtr

8.1.1.6 uds_transport::UdsMessagePtr

[SWS_DM_00303] ⌈ The type alias ara::diag::uds_transport::UdsMessagePtr is defined in Table 8.6. ⌋*()*

Kind:	type alias
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef std::unique_ptr<UdsMessage, std::function<void(UdsMessage*)»>
Syntax:	using ara::diag::uds_transport::UdsMessagePtr = std::unique_ptr<UdsMessage, std::function<void(UdsMessage*)»>;
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	This is the unique_ptr for UdsMessages containing a custom deleter as provided by the generic/core DM part towards the UdsTransportLayer-Plugin. How the exact typedef for UdsMessagePtr looks like, is up to the DM product vendor. I.e. how f.i. the deleter signature looks like ... basically the minimal agreement is: UdsMessagePtr shall behave like a std::unique_ptr<UdsMessage>!
Notes:	How the exact typedef for UdsMessagePtr looks like, is up to the DM product vendor. I.e. how f.i. the deleter signature looks like ... basically the minimal agreement is: UdsMessagePtr shall behave like a std::unique_ptr<UdsMessage>!

Table 8.6: type alias ara::diag::uds_transport::UdsMessagePtr

8.1.1.7 uds_transport::UdsTransportProtocolHandlerID

[SWS_DM_00336] ⌈ The type alias ara::diag::uds_transport::UdsTransportProtocolHandlerID is defined in Table 8.7. ⌋*()*

Kind:	type alias
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef uint8_t
Syntax:	using ara::diag::uds_transport::UdsTransportProtocolHandlerID = uint8_t;
Header file:	#include "ara/diag/uds_transport/protocol_types.h"
Description:	UdsTransportProtocolHandler are flexible "plugins", which need an identification.

Table 8.7: type alias ara::diag::uds_transport::UdsTransportProtocolHandlerID

8.1.2 UdsMessage Class

[SWS_DM_00291] ⌈ The class ara::diag::uds_transport::UdsMessage is defined in Table 8.8. ⌋*()*

Kind:	class
Base class:	None
Syntax:	`class UdsMessage`
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	class represents an UDS message exchanged between DM generic core (UdsTransport ProtocolMgr) and a specific implementation of UdsTransportProtocolHandler on diagnostic request reception path or diagnostic response transmission path. UdsMessage provides the storage for UDS requests/responses. Instances of UdsMessage (with optimized resource allocation) are only created by DM generic core. UdsTransport ProtocolHandler read/write on it.

Table 8.8: class ara::diag::uds_transport::UdsMessage

8.1.2.1 Types

8.1.2.1.1 uds_transport::UsdMessage::Address

[SWS_DM_00293] ⌈ The type alias ara::diag::uds_transport::UdsMessage::Address is defined in Table 8.9. ⌋*()*

Kind:	type alias
Scope:	class ara::diag::uds_transport::UdsMessage
Derived from:	uint16_t
Syntax:	using ara::diag::uds_transport::UdsMessage::Address = uint16_t;
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	type for UDS source and target addresses

Table 8.9: type alias ara::diag::uds_transport::UdsMessage::Address

8.1.2.1.2 uds_transport::UsdMessage::MetaInfoMap

[SWS_DM_00294] ⌈ The type alias ara::diag::uds_transport::UdsMessage::MetaInfo Map is defined in Table 8.10. ⌋*()*

Kind:	type alias
Scope:	class ara::diag::uds_transport::UdsMessage
Derived from:	ara::core::Map<ara::core::String, ara::core::String>
Syntax:	using ara::diag::uds_transport::UdsMessage::MetaInfoMap = ara::core::Map<ara::core::String, ara::core::String>;
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	Type for the meta information attached to a UdsMessage.

Table 8.10: type alias ara::diag::uds_transport::UdsMessage::MetaInfoMap

8.1.2.1.3 uds_transport::UsdMessage::TargetAddressType

[SWS_DM_00296] ⌈ The enum ara::diag::uds_transport::UdsMessage::TargetAddressType is defined in Table 8.11. ⌋()

Kind:	enum	
Values:	kPhysical= 0	–
	kFunctional= 1	–
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	type of target address in UdsMessage	

Table 8.11: enum ara::diag::uds_transport::UdsMessage::TargetAddressType

8.1.2.1.4 uds_transport::UdsMessage::payload

[SWS_DM_09028] ⌈ The variable ara::diag::uds_transport::UdsMessage::payload_ is defined in Table 8.12. ⌋()

Kind:	variable
Type:	uds_transport::ByteVector
Scope:	class ara::diag::uds_transport::UdsMessage
Syntax:	uds_transport::ByteVector ara::diag::uds_transport::UdsMessage::payload_;
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	Payload of the uds message.

Table 8.12: variable ara::diag::uds_transport::UdsMessage::payload_

8.1.2.2 Methods

8.1.2.2.1 uds_transport::UdsMessage::UdsMessage

[SWS_DM_09012] ⌈ The function ara::diag::uds_transport::UdsMessage::UdsMessage is defined in Table 8.13. ⌋()

Symbol:	ara::diag::uds_transport::UdsMessage::UdsMessage()
Kind:	function
Scope:	class ara::diag::uds_transport::UdsMessage
Visibility:	protected
Syntax:	`inline UdsMessage ();`
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	non public default ctor. The default ctor is protected as we want to forbid, that UdsTransport Protocol handlers do create UdsMessages on its own! Only DM is allowed to create and hands over UdsMessagePtrs to UdsTransportProtocolHandler.

Table 8.13: function ara::diag::uds_transport::UdsMessage::UdsMessage

8.1.2.2.2 uds_transport::UdsMessage::~UdsMessage

[SWS_DM_09010] ⌈ The function ara::diag::uds_transport::UdsMessage::~UdsMessage is defined in Table 8.14. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsMessage::~UdsMessage()
Kind:	function
Scope:	class ara::diag::uds_transport::UdsMessage
Syntax:	`inline virtual ~UdsMessage ();`
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	Destructing the uds message.

Table 8.14: function ara::diag::uds_transport::UdsMessage::~UdsMessage

8.1.2.2.3 uds_transport::UdsMessage::AddMetaInfo

[SWS_DM_00302] ⌈ The function ara::diag::uds_transport::UdsMessage::AddMetaInfo is defined in Table 8.15. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsMessage::AddMetaInfo(std::shared_ptr< const MetaInfoMap > meta_info)	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsMessage	
Syntax:	`virtual void AddMetaInfo (std::shared_ptr< const MetaInfoMap > meta_info);`	
Parameters (in):	meta_info	meta information relevant for UdsMessage
Return value:	None	
Thread Safety:	unsafe	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	add new metaInfo to this message.	
Notes:	typically called by the transport plugin to add channel specific meta-info. (see SWS - there are already predefined meta-info keys for DoIP....)	

Table 8.15: function ara::diag::uds_transport::UdsMessage::AddMetaInfo

8.1.2.2.4 uds_transport::UdsMessage::GetPayload

[SWS_DM_00300] ⌈ The function ara::diag::uds_transport::UdsMessage::GetPayload is defined in Table 8.16. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsMessage::GetPayload()	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsMessage	
Syntax:	`inline virtual const uds_transport::ByteVector& GetPayload () const ;`	
Return value:	const uds_transport::ByteVector &	The entire payload (A_Data)
Thread Safety:	unsafe	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Get the UDS message data starting with the SID (A_Data as per ISO) The entire payload (A_Data) marked as "unsafe" with regard to threadsafety as implementation is allowed to do ressource allocation of buffer in the context of this call.	
Notes:	marked as "unsafe" with regard to threadsafety as implementation is allowed to do ressource allocation of buffer in the context of this call.	

Table 8.16: function ara::diag::uds_transport::UdsMessage::GetPayload

[SWS_DM_00301] ⌈ The function ara::diag::uds_transport::UdsMessage::GetPayload is defined in Table 8.17. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsMessage::GetPayload()	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsMessage	
Syntax:	`inline virtual uds_transport::ByteVector& GetPayload ();`	
Return value:	uds_transport::ByteVector &	payload of the UDSMessage starting from SID.
Thread Safety:	unsafe	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	return the underlying buffer for write access. needed by UdsTransportProtocolHandler impl. to fill the UdsMessage with data in RX path. I.e. UdsTransportProtocolHandler impl. gets the UdsMessage instance from call to UdsTransport ProtocolMgr::IndicateMessage() and then calls this method on it and write into returned uds_transport::ByteVector. payload of the UDSMessage starting from SID. marked as "unsafe" with regard to threadsafety as implementation is allowed to do ressource allocation of buffer in the context of this call.	
Notes:	needed by UdsTransportProtocolHandler impl. to fill the UdsMessage with data in RX path. I.e. UdsTransportProtocolHandler impl. gets the UdsMessage instance from call to UdsTransport ProtocolMgr::IndicateMessage() and then calls this method on it and write into returned uds_transport::ByteVector. marked as "unsafe" with regard to threadsafety as implementation is allowed to do ressource allocation of buffer in the context of this call.	

Table 8.17: function ara::diag::uds_transport::UdsMessage::GetPayload

8.1.2.2.5 uds_transport::UdsMessage::GetSa

[SWS_DM_00297] ⌈ The function ara::diag::uds_transport::UdsMessage::GetSa is defined in Table 8.18. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsMessage::GetSa()	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsMessage	
Syntax:	`virtual Address GetSa () const noexcept;`	
Return value:	Address	The source address of the uds message.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Get the source address of the uds message. The source address of the uds message.	

Table 8.18: function ara::diag::uds_transport::UdsMessage::GetSa

8.1.2.2.6 uds_transport::UdsMessage::GetTa

[SWS_DM_00298] ⌈ The function ara::diag::uds_transport::UdsMessage::GetTa is defined in Table 8.19. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsMessage::GetTa()	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsMessage	
Syntax:	`virtual Address GetTa () const noexcept;`	
Return value:	Address	The target address of the uds message.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Get the target address of the uds message. The target address of the uds message.	

Table 8.19: function ara::diag::uds_transport::UdsMessage::GetTa

8.1.2.2.7 uds_transport::UdsMessage::GetTaType

[SWS_DM_00299] ⌈ The function ara::diag::uds_transport::UdsMessage::GetTaType is defined in Table 8.20. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsMessage::GetTaType()	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsMessage	
Syntax:	`virtual TargetAddressType GetTaType () const noexcept;`	
Return value:	TargetAddressType	The target address type of the uds message.

▽

△

Exception Safety:	noexcept
Thread Safety:	reentrant
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	Get the target address type (phys/func) of the uds message. The target address type of the uds message.

Table 8.20: function ara::diag::uds_transport::UdsMessage::GetTaType

8.1.3 UdsTransportProtocolHandler Class

[SWS_DM_00315] ⌈ The class ara::diag::uds_transport::UdsTransportProtocolHandler is defined in Table 8.21. ⌋*()*

Kind:	class	
Base class:	None	
Syntax:	`class UdsTransportProtocolHandler`	
Protected fields:	`UdsTransportProtocolMgr & transportprotocol_manager_`	The UdsTransportProtocolMgr used/provided by the DM/DCM.
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Abstract Class, which a specific UDS Transport Protocol (plugin) shall subclass.	

Table 8.21: class ara::diag::uds_transport::UdsTransportProtocolHandler

8.1.3.1 Types

8.1.3.1.1 uds_transport::UdsTransportProtocolHandler::InitializationResult

[SWS_DM_09017] ⌈ The enum ara::diag::uds_transport::UdsTransportProtocolHandler::InitializationResult is defined in Table 8.22. ⌋*()*

Kind:	enum	
Values:	kInitializeOk= 0	–
	kInitializeFailed= 1	–
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Result of Initialize handler.	

Table 8.22: enum ara::diag::uds_transport::UdsTransportProtocolHandler::InitializationResult

8.1.3.2 Methods

8.1.3.2.1 uds_transport::UdsTransportProtocolHandler::UdsTransportProtocolHandler

[SWS_DM_09015] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::UdsTransportProtocolHandler is defined in Table 8.23. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::UdsTransportProtocolHandler(const UdsTransportProtocolHandlerID handler_id, UdsTransportProtocolMgr &transport_protocol_mgr)	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler	
Syntax:	`explicit inline UdsTransportProtocolHandler (const UdsTransport ProtocolHandlerID handler_id, UdsTransportProtocolMgr &transport_protocol_mgr);`	
Parameters (in):	handler_id	the handler ID used by DM to identify this handler. This is just a number/identification given by the DM core when instantiating a UdsTransportProtocol Handler instance to be able to distinguish it from other handler-plugins or built-in UdsTransport ProtocolHandler implementations.
	transport_protocol_mgr	reference to UdsTransportProtocolMgr owned by this DM, with which UdsTransportProtocolHandler instance shall interact.
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Constructor of UdsTransportProtocolHandler.	

Table 8.23: function ara::diag::uds_transport::UdsTransportProtocolHandler::UdsTransportProtocolHandler

8.1.3.2.2 uds_transport::UdsTransportProtocolHandler::~UdsTransport

[SWS_DM_09016] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::~UdsTransportProtocolHandler is defined in Table 8.24. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::~UdsTransportProtocolHandler()
Kind:	function
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler
Syntax:	`inline virtual ~UdsTransportProtocolHandler ();`
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"
Description:	Destructor of UdsTransportProtocolHandler.

Table 8.24: function ara::diag::uds_transport::UdsTransportProtocolHandler::~UdsTransportProtocolHandler

8.1.3.2.3 uds_transport::UdsTransportProtocolHandler::GetHandlerID

[SWS_DM_00325] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::GetHandlerID is defined in Table 8.25. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::GetHandlerID()	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler	
Syntax:	`inline virtual UdsTransportProtocolHandlerID GetHandlerID () const ;`	
Return value:	UdsTransportProtocolHandlerID	UdsTransportProtocolHandlerID.
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Return the UdsTransportProtocolHandlerID, which was given to the implementation during construction (ctor call). UdsTransportProtocolHandlerID.	

Table 8.25: function ara::diag::uds_transport::UdsTransportProtocolHandler::GetHandlerID

8.1.3.2.4 uds_transport::UdsTransportProtocolHandler::Initialize

[SWS_DM_00319] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::Initialize is defined in Table 8.26. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::Initialize()	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler	
Syntax:	`virtual InitializationResult Initialize ()=0;`	
Return value:	InitializationResult	kInitializeOk if initialization was successful, else k InitializeFailed.
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Initializes handler. Must be called before Start(). The idea is to have "initialization" of handler-plugin separated from its ctor. kInitializeOk if initialization was successful, else kInitializeFailed.	

Table 8.26: function ara::diag::uds_transport::UdsTransportProtocolHandler::Initialize

8.1.3.2.5 uds_transport::UdsTransportProtocolHandler::NotifyReestablishment

[SWS_DM_00326] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::NotifyReestablishment is defined in Table 8.27. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::NotifyReestablishment(ChannelID channel_id)
Kind:	function
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler
Syntax:	`virtual bool NotifyReestablishment (ChannelID channel_id)=0;`

▽

△

Parameters (in):	channel_id	channelID, whose re-establishment shall be notified to UdsTransportProtocolMgr
Return value:	bool	true if notification request is accepted and can be fulfilled.
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Tells the UdsTransportProtocolHandler, that it shall notify the DM core via UdsTransportProtocol Mgr::ChannelReestablished()) if the given channel has been re-established after next Uds TransportProtocolHandler::Start(). The main purpose of this method is to allow DM to provide an ECU-Reset (0x11 service), with configuration option "Pos. response AFTER reset". In this scenario the request for 0x11 will be received on a certain channel with identifying tuple <p_x, c_y> (GlobalChannelIdentifier). Then the ECU-Reset takes place and after ECU-Restart all UdsProtocolHandlers/plugins get restarted via call to UdsTransportProtocolHandler::Start(). Now there are two expectations, when this method has been called before and returned "true": IF the same remote client connects to the UdsProtocolHandler, it shall get a channel identification with the same identifying tuple <p_x, c_y> as last time.it shall call UdsTransportProtocolMgr::Channel Reestablished(GlobalChannelIdentifier<p_x, c_y>)	
Notes:	: IF the underlying network layer of the UdsTransportProtocolHandler isn't really connection based (e.g. a UDP based protocol), then the UdsTransportProtocolHandler shall call Uds TransportProtocolMgr::ChannelReestablished() after UdsTransportProtocolHandler::Start() as soon as it detects/assumes that the remote client/tester will be reachable again. : The detection/decision, whether the "same" client reconnects as before is an UdsProtocol Handler implementation specific decision. The general expectation is: If the channel is set up from exactly the same remote network-endpoint, it typically shall be given the same channelID (c_y part of the tuple). To support this functionality the implementation at least has to store non-volatile, that this notification has to be done. Further it might be needed to store some additional connection specific info non-volatile to make sure, that the same channelID (c_y part of the tuple) can be reassigned. This is the case if the mapping of protocol specific channel info -> channelID isn't a stable bijective mapping! Small example: The underlying network protocol, which UdsProtocolHandler implements is based on TCP. At the point in time, where the 0x11 SI request is received on channel identified by <p_x, c_y> the DM calls NotifyReestablishment() on this channelID. Now the implementation of UdsProtocolHandler stores non-volatile in the context of this call: the NetworkEndpoint (IP-address and port number) of the channelthe NetworkEndpoint (IP-address and port number) of the local port (because in this example, the UdsTransportProtocolHandler listens on/supports different ports)the channelID (c_y part) it has currently assigned. After restart this channelID only shall be reused for a channel with exactly the same NetworkEndpoint addresses as stored non-volatile. If this channelID then gets reassigned, then UdsTransportProtocolMgr::ChannelReestablished() has to be called.	

Table 8.27: function ara::diag::uds_transport::UdsTransportProtocolHandler::Notify Reestablishment

8.1.3.2.6 uds_transport::UdsTransportProtocolHandler::Start

[SWS_DM_00322] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::Start is defined in Table 8.28. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::Start()
Kind:	function
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler
Syntax:	`virtual void Start ()=0;`

▽

△

Return value:	None
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"
Description:	Start processing the implemented Uds Transport Protocol. The implementation shall call its superclass Start() method as there might be some stack specific implementation. Implementation shall be asynchronous as DM might start many/different UdsTransportProtocolHandler in parallel and strong serialization of all those starts just unnecessarily slows down DM startup.

Table 8.28: function ara::diag::uds_transport::UdsTransportProtocolHandler::Start

8.1.3.2.7 uds_transport::UdsTransportProtocolHandler::Stop

[SWS_DM_00323] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::Stop is defined in Table 8.29. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::Stop()
Kind:	function
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler
Syntax:	`virtual void Stop ()=0;`
Return value:	None
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"
Description:	Method to indicate that this UdsTransportProtocolHandler should terminate. If UdsTransportProtocolHandler has stopped, it shall call UdsTransportProtocolMgr::Handler Stopped(UdsTransportProtocolHandlerID) After return from Stop(), the handler-plugin shall NOT call to UdsTransportProtocolMgr with any other method but UdsTransportProtocolMgr::HandlerStopped()

Table 8.29: function ara::diag::uds_transport::UdsTransportProtocolHandler::Stop

8.1.3.2.8 uds_transport::UdsTransportProtocolHandler::Transmit

[SWS_DM_00327] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolHandler::Transmit is defined in Table 8.30. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::Transmit(UdsMessageConstPtr message, ChannelID channel_id)
Kind:	function
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler
Syntax:	`virtual void Transmit (UdsMessageConstPtr message, ChannelID channel_id)=0;`

▽

△

Parameters (in):	message	The message to be transmitted as a Uds Message::Ptr (unique_ptr style). UdsTransport ProtocolHandler has to give back this Uds Message::Ptr via UdsTransportProtocol Mgr::TransmitConfirmation() to signal, that it is done with this message.
	channel_id	identification of channel on which to transmit.
Return value:	None	
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Transmit a Uds message via the underlying Uds Transport Protocol channel. This transmit API covers T_Data.req of ISO 14229-2 Figure 2.	

Table 8.30: function ara::diag::uds_transport::UdsTransportProtocolHandler::Transmit

8.1.4 UdsTransportProtocolMgr Class

[SWS_DM_00306] ⌈ The class ara::diag::uds_transport::UdsTransportProtocolMgr is defined in Table 8.31. ⌋*()*

Kind:	class
Base class:	None
Syntax:	`class UdsTransportProtocolMgr`
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"
Description:	

Table 8.31: class ara::diag::uds_transport::UdsTransportProtocolMgr

8.1.4.1 Types

8.1.4.1.1 uds_transport::UdsTransportProtocolMgr::GlobalChannelIdentifier

[SWS_DM_09021] ⌈ The type alias ara::diag::uds_transport::UdsTransportProtocol Mgr::GlobalChannelIdentifier is defined in Table 8.32. ⌋*()*

Kind:	type alias
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr
Derived from:	std::tuple<UdsTransportProtocolHandlerID, ChannelID>
Syntax:	using ara::diag::uds_transport::UdsTransportProtocolMgr::GlobalChannelIdentifier = std::tuple<UdsTransportProtocolHandlerID, ChannelID>;
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"
Description:	Type of tuple to pack UdsTransportProtocolHandlerID and ChannelID together, to form a global unique (among all used UdsTransportProtocolHandlers within DM) identifier of a UdsTransport Protocol channel.

Table 8.32: type alias ara::diag::uds_transport::UdsTransportProtocolMgr::GlobalChannelIdentifier

8.1.4.1.2 uds_transport::UdsTransportProtocolMgr::IndicationResult

[SWS_DM_00384] ⌈ The enum ara::diag::uds_transport::UdsTransportProtocolMgr::IndicationResult is defined in Table 8.33. ⌋()

Kind:	enum	
Values:	kIndicationOk= 0	–
	kIndicationOccupied= 1	–
	kIndicationOverflow= 2	–
	kIndicationUnknownTargetAddress= 3	–
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:		

Table 8.33: enum ara::diag::uds_transport::UdsTransportProtocolMgr::IndicationResult

8.1.4.1.3 uds_transport::UdsTransportProtocolMgr::TransmissionResult

[SWS_DM_00307] ⌈ The enum ara::diag::uds_transport::UdsTransportProtocolMgr::TransmissionResult is defined in Table 8.34. ⌋()

Kind:	enum	
Values:	kTransmitOk= 0	–
	kTransmitFailed= 1	–
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:		

Table 8.34: enum ara::diag::uds_transport::UdsTransportProtocolMgr::TransmissionResult

8.1.4.2 Methods

8.1.4.2.1 uds_transport::UdsTransportProtocolMgr::ChannelReestablished

[SWS_DM_00313] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolMgr::ChannelReestablished is defined in Table 8.35. ⌋()

Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::ChannelReestablished(GlobalChannelIdentifier global_channel_id)	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Syntax:	`virtual void ChannelReestablished (GlobalChannelIdentifier global_channel_id)=0;`	
Parameters (in):	global_channel_id	transport protocol channel, which is available again.

▽

△

Return value:	None
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"
Description:	notification call from the given transport channel, that it has been reestablished since the last (Re)Start from the UdsTransportProtocolHandler to which this channel belongs. To activate this notification a previous call to UdsTransportProtocolHandler::NotifyReestablishment() has to be done. See further documentation at UdsTransportProtocolHandler::NotifyReestablishment().

Table 8.35: function ara::diag::uds_transport::UdsTransportProtocolMgr::Channel Reestablished

8.1.4.2.2 uds_transport::UdsTransportProtocolMgr::HandleMessage

[SWS_DM_00311] ⌈ The function ara::diag::uds_transport::UdsTransportProtocol Mgr::HandleMessage is defined in Table 8.36. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::HandleMessage(UdsMessagePtr message)	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Syntax:	`virtual void HandleMessage (UdsMessagePtr message)=0;`	
Parameters (in):	message	The Uds message ptr (unique_ptr semantics) with the request. Ownership of the UdsMessage is given back to the generic DM core here.
Return value:	None	
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:	Hands over a valid received Uds message (currently this is only a request type) from transport layer to session layer. It corresponds to T_Data.ind of Figure 2 from ISO 14229-2. The behavior is asynchronously. I.e. the UdsMessage is handed over to Session Layer and it is expected, that it "instantly" returns, which means, that real processing of the message shall be done asynchronously!	

Table 8.36: function ara::diag::uds_transport::UdsTransportProtocolMgr::HandleMessage

8.1.4.2.3 uds_transport::UdsTransportProtocolMgr::HandlerStopped

[SWS_DM_00314] ⌈ The function ara::diag::uds_transport::UdsTransportProtocol Mgr::HandlerStopped is defined in Table 8.37. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::HandlerStopped(UdsTransportProtocol HandlerID handler_id)
Kind:	function
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr
Syntax:	`virtual void HandlerStopped (UdsTransportProtocolHandlerID handler_id)=0;`

▽

△

Parameters (in):	handler_id	indication, which plugin stopped.
Return value:	None	
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:	notification from handler, that it has stopped now (e.g. closed down network connections, freed resources, etc...) This callback is expected as a reaction from handler to a call to UdsTransportProtocol Handler::Stop.	

Table 8.37: function ara::diag::uds_transport::UdsTransportProtocolMgr::Handler Stopped

8.1.4.2.4 uds_transport::UdsTransportProtocolMgr::IndicateMessage

[SWS_DM_00309] ⌈ The function ara::diag::uds_transport::UdsTransportProtocol Mgr::IndicateMessage is defined in Table 8.38. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::IndicateMessage(UdsMessage::Address source_addr, UdsMessage::Address target_addr, UdsMessage::TargetAddressType type, GlobalChannelIdentifier global_channel_id, std::size_t size, Priority priority, ProtocolKind protocol_kind)	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Syntax:	`virtual std::pair<IndicationResult, UdsMessagePtr> IndicateMessage (UdsMessage::Address source_addr, UdsMessage::Address target_addr, UdsMessage::TargetAddressType type, GlobalChannelIdentifier global_channel_id, std::size_t size, Priority priority, ProtocolKind protocol_kind)=0;`	
Parameters (in):	source_addr	UDS source address of message
	target_addr	UDS target address of message
	type	indication whether its is phys/func request
	global_channel_id	transport protocol channel on which message start happened
	size	size in bytes of the UdsMessage starting from SID.
	priority	the priority of the given message, used for prioritization of conversations.
	protocol_kind	identifier of protocol kind associated to message
Return value:	std::pair< IndicationResult, Uds MessagePtr >	Pair of IndicationResult and a pointer to Uds Message owned/created by DM core and returned to the handler to get filled.
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:	Indicates a message start. This is an interface, which is just served/called by UdsTransport ProtocolHandlers, which return true from UdsTransportProtocolHandlers::isStartOfMessage IndicationSupported().	

Table 8.38: function ara::diag::uds_transport::UdsTransportProtocolMgr::IndicateMessage

8.1.4.2.5 uds_transport::UdsTransportProtocolMgr::NotifyMessageFailure

[SWS_DM_00310] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolMgr::NotifyMessageFailure is defined in Table 8.39. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::NotifyMessageFailure(UdsMessagePtr message)	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Syntax:	`virtual void NotifyMessageFailure (UdsMessagePtr message)=0;`	
Parameters (in):	message	the pointer to UdsMessage handed back over to the session layer.
Return value:	None	
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:	Indicates, that the message indicated via IndicateMessage() has failure and will not lead to a final HandleMessage() call.	

Table 8.39: function ara::diag::uds_transport::UdsTransportProtocolMgr::NotifyMessageFailure

8.1.4.2.6 uds_transport::UdsTransportProtocolMgr::TransmitConfirmation

[SWS_DM_00312] ⌈ The function ara::diag::uds_transport::UdsTransportProtocolMgr::TransmitConfirmation is defined in Table 8.40. ⌋*()*

Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::TransmitConfirmation(UdsMessageConstPtr message, TransmissionResult result)	
Kind:	function	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Syntax:	`virtual void TransmitConfirmation (UdsMessageConstPtr message, TransmissionResult result)=0;`	
Parameters (in):	message	for which message (created in IndicateMessage()) this is the confirmation.
	result	Result of transmission. In case UDS message could be transmitted on network layer: kTransmitOk), k TransmitFailed else.
Return value:	None	
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:	notification about the outcome of a transmit request called by core DM at the handler via Uds TransportProtocolHandler::Transmit This transmit API covers T_Data.con of ISO 14229-2 Figure 2.	

Table 8.40: function ara::diag::uds_transport::UdsTransportProtocolMgr::TransmitConfirmation

8.1.5 Sequence Diagramms of UDS Transport Layer Interaction

8.1.5.1 Lifecycle

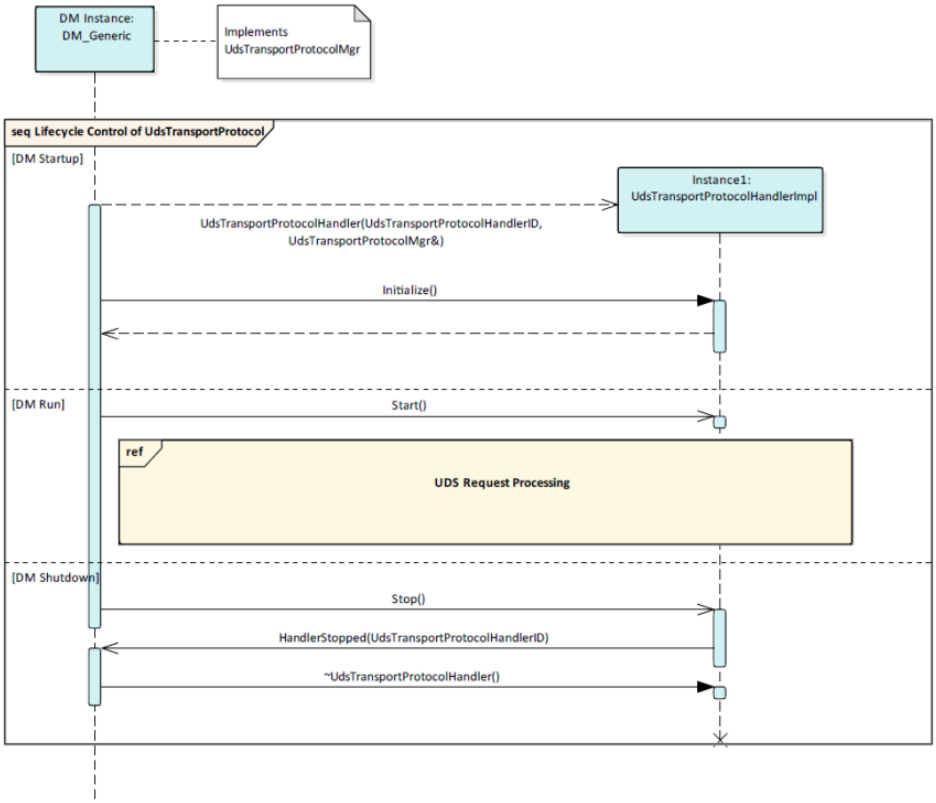

Figure 8.1: UDS Transport Lifecycle

8.1.5.2 UDS Request Processing

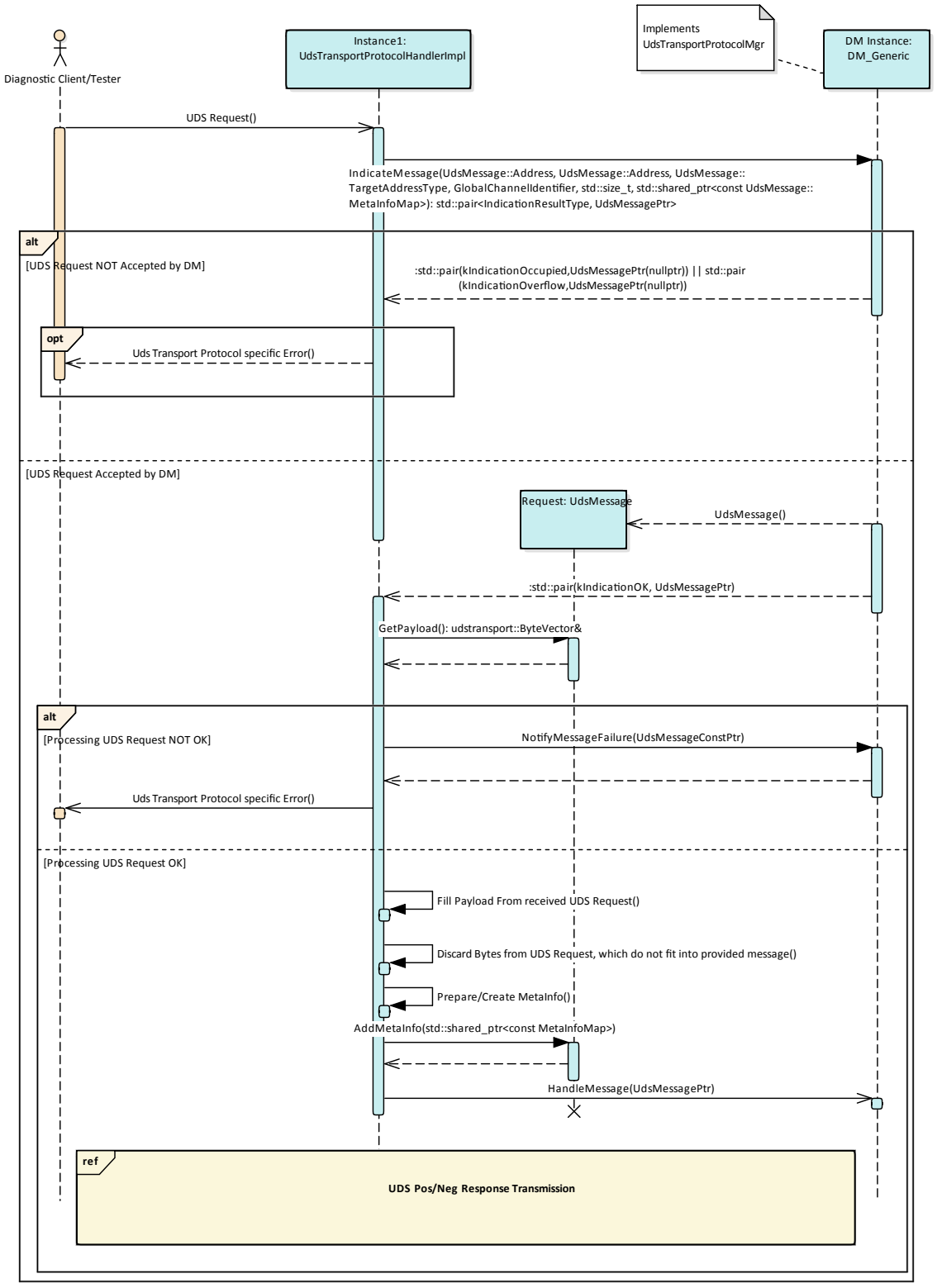

Figure 8.2: UDS Transport Request Processing

8.1.5.3 UDS Response Transmission

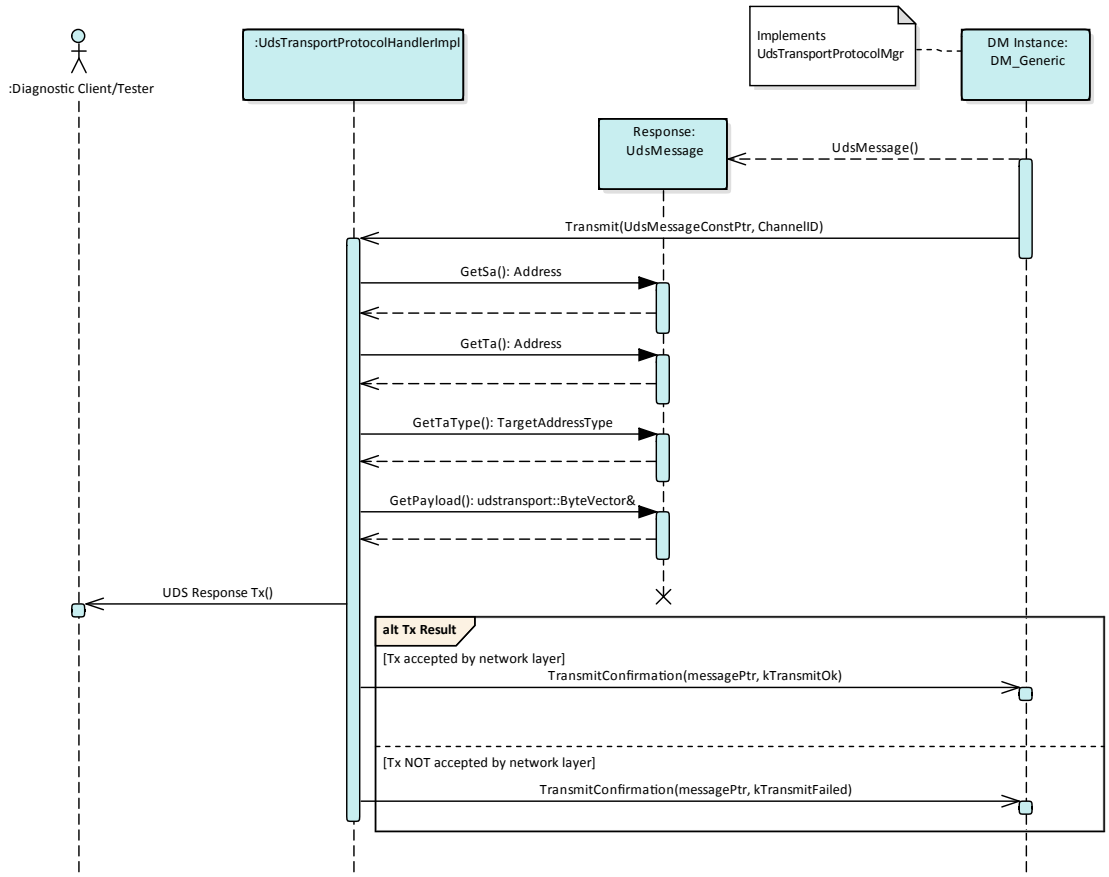

Figure 8.3: UDS Response Transmission

8.1.5.4 Channel Reestablishment

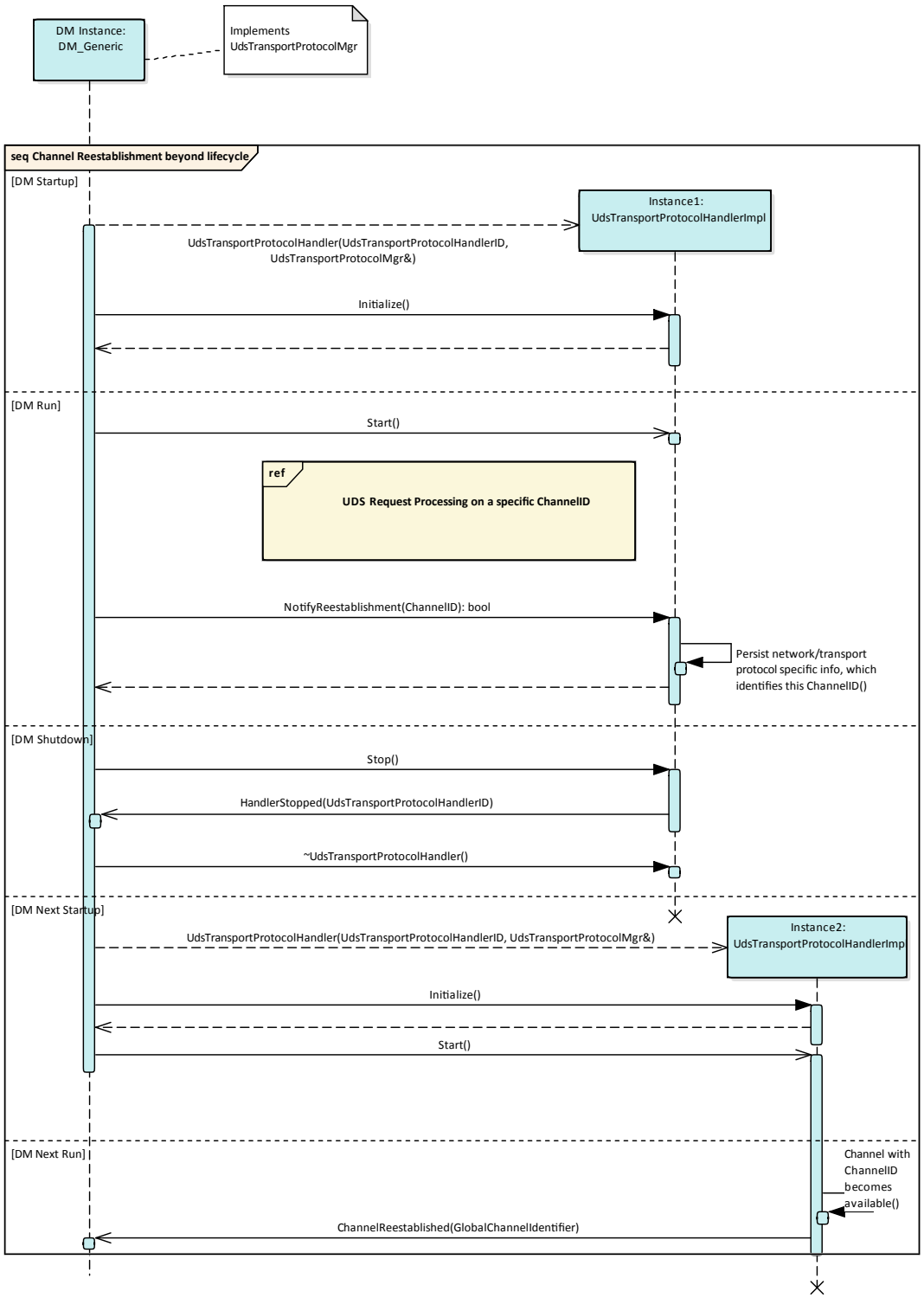

Figure 8.4: UDS Transport Channel Reestablishment

9 Service Interfaces

This chapter lists all provided and required service interfaces of the `Diagnostic Management`.

All variations defined in the following tables are described by the ARMQL (AUTOSAR Model Query Language) as defined in GST [14]. It allows in a clear and executable way to describe how interface variations are transformed to real interfaces dependent on a specific configuration. This language consists of FOR, LET and WHERE blocks. The **FOR** block iterates over a sequence, the **LET** block is used to give names to expressions and functions and the **WHERE** block specifies a condition to filter the items. Furthermore names defined at a higher level element are also visible at lower level elements.

9.1 Type definitions

This chapter lists all types provided by the DM.

9.1.1 Diagnostic service management

9.1.1.1 ActivityStatusType

Name	ActivityStatusType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Possible states of an diagnostic conversation.	
Range / Symbol	***Limit***	***Description***
kActive	0x00	Currently active; i.e. request is curretnly processed or non-default session is active.
kInactive	0x01	Currently not active

Table 9.1: Implementation Data Type - ActivityStatusType

9.1.1.2 DiagnosticSessionType

Name	DiagnosticSessionType
Kind	TYPE_REFERENCE
Derived from	`string`
Description	Represents the Diagnostic Session.

Table 9.2: Implementation Data Type - DiagnosticSessionType

9.1.1.3 DiagnosticSecurityLevelType

Name	DiagnosticSecurityLevelType
Kind	TYPE_REFERENCE
Derived from	`string`
Description	Represents the Diagnostic Security Level.

Table 9.3: Implementation Data Type - DiagnosticSecurityLevelType

9.1.1.4 DiagnosticConversationIdentifierType

Name	DiagnosticConversationIdentifierType
Kind	STRUCTURE
Subelements	diagnosticProtocolKind `string` diagnsoticServerAddress `UdsAddressType` testerAddress `UdsAddressType`
Derived from	-
Description	Characterizes an ongoing Diagnostic Conversation.

Table 9.4: Implementation Data Type - DiagnosticConversationIdentifierType

9.1.1.5 UdsAddressType

Name	UdsAddressType
Kind	TYPE_REFERENCE
Derived from	`uint16_t`
Description	Represents the UDS Address as defined in ISO-14229-1.

Table 9.5: Implementation Data Type - UdsAddressType

9.1.1.6 ByteVectorType

Name	ByteVectorType
Kind	VECTOR
Subelements	`uint8_t`
Derived from	-
Description	DataArrayType

Table 9.6: Implementation Data Type - ByteVectorType

9.1.1.7 MetaInfoType

Name	MetaInfoType
Kind	ASSOCIATIVE_MAP
Subelements	key `string` value `string`
Derived from	-
Description	Meta-Inf map, which contains key-value pairs of MetaInfoKeyType, MetaInfoValueType.

Table 9.7: Implementation Data Type - MetaInfoType

9.1.1.7.1 Standardized key values

Key	Format	Description
kSA	[0-9A-Fa-f]{2,4}	UDS Source Address from which the diagnostic request has been sent. The value in the MetaInf Map for this key, will be a stringified form of UDS source address in hex. For example tester SA of decimal 240 will have the stringified value "F0"
kTA	[0-9A-Fa-f]{2,4}	UDS Target Address to which the diagnostic request has been sent. The value in the MetaInf Map for this key, will be a stringified form of UDS source address in hex. For example TA of decimal 59 will have the stringified value "3B"
kTAType	enumeration	Indicator whether request is functional or physical addressed. The value in the MetaInf Map for this key, will be either "PHYS" or "FUNC"
kSuppPosResponse	boolean	Key for the flag, whether positive response shall be suppressed for current request. The value in the MetaInf Map for this key, will be the either "TRUE" or "FLASE"
kDoIPLocalIP	IPv4 or IPv6	Key for the local IP address on which the current request gets received in case of DoIP is used as UDS transport layer (this might be of interest in case the ECU is multihomed and could receive diagnostic requests via DoIP on different interfaces). The value in the MetaInf Map for this key, will be either a string in IPv4 address notation (decimal representation of address parts separated with ".") or a string in IPv6 notation (hexadecimal representation of address parts separated with ":" according to section 2.2 of RFC 4291
kDoIPLocalPort	[0-9]{1,5}	Key for the local port number on which the current request gets received in case of DoIP is used as UDS transport layer. The value in the MetaInf Map for this key, will be the stringified decimal representation of the port number.
kDoIPRemoteIP	IPv4 or IPv6	Key for the remote IP address on which the current request gets received in case of DoIP is used as UDS transport layer. The value in the MetaInf Map for this key, will be either a string in IPv4 address notation (decimal representation of address parts separated with ".") or a string in IPv6 notation (hexadecimal representation of address parts separated with ":" according to section 2.2 of RFC 4291)

kDoIPRemotePort	[0-9]{1,5}	Key for the remote port number on which the current request gets received in case of DoIP is used as UDS transport layer. The value in the MetaInf Map for this key, will be the stringified decimal representation of the port number.
kConversationInstanceId		Instance identification of DiagnosticConversation_{SoftwareCluster}_{Number} ServiceInterface, which could be used directly in FindService()
kContext	enumeration	DiagnosticCommunication; FaultMemory; DoIP
kDtc	[0-9A-Fa-f]{1,6}	DTC number for which this interface is triggered; only if kContext == FaultMemory

Table 9.8: Standardized key values in `MetaInfoType`

9.1.1.8 KeyCompareResultType

Name	KeyCompareResultType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description		
Range / Symbol	***Limit***	***Description***
kKeyValid	0x00	Key is valid
kKeyInvalid	0x01	Key is invalid

Table 9.9: Implementation Data Type - KeyCompareResultType

9.1.1.9 ControlDtcStatusType

Name	ControlDtcStatusType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Type for ControlDTCStatus status	
Range / Symbol	***Limit***	***Description***
kDTCSettingOn	0x00	Updating of diagnostic trouble code status bits is under normal operating conditions
kDTCSettingOff	0x01	Updating of diagnostic trouble code status bits is stopped

Table 9.10: Implementation Data Type - ControlDtcStatusType

9.1.1.10 CommunicationControlStatusType

Name	ComCtrlRequestParamsType
Kind	STRUCTURE
Subelements	controlType `uint8_t` communicationType `uint8_t` nodeIdentificationNumber `uint16_t`

▽

△

Derived from	-
Description	ComCtrlRequestParamsType is a structure, which holds all parameters of an UDS 0x28 communicationControl request.

Table 9.11: Implementation Data Type - ComCtrlRequestParamsType

9.1.1.11 ConfirmationStatusType

Name	ConfirmationStatusType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Type used in the method confirmed for the status of the service processing	
Range / Symbol	***Limit***	***Description***
kResPosOk	0x00	Positive response has been sent out successfully
kResPosNotOk	0x01	Positive response has not been sent out successfully
kResNegOk	0x02	Negative response has been sent out successfull
kResNegNotOk	0x03	Negative response has not been sent out successfully
kResPosSuppressed	0x04	Positive answer suppressed
kResNegSuppressed	0x05	Negative answer suppressed
kCanceled	0x06	Processing is canceled
kNoProcessingNoResponse	0x07	Processing rejected in Validation

Table 9.12: Implementation Data Type - ConfirmationStatusType

9.1.1.12 StateType

Name	StateType
Kind	TYPE_REFERENCE
Derived from	`bool`
Description	boolean

Table 9.13: Implementation Data Type - StateType

9.1.1.13 SIDType

Name	SIDType
Kind	TYPE_REFERENCE
Derived from	`uint8_t`
Description	SIDType

Table 9.14: Implementation Data Type - SIDType

9.1.1.14 VINType

Name	VINType
Kind	ARRAY
Subelements	`uint8_t`
Derived from	-
Description	17 byte array of uint8 representing the VIN

Table 9.15: Implementation Data Type - VINType

9.1.1.15 ExecutionTypeType

Name	ExecutionTypeType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Type of execution of the restart.	
Range / Symbol	***Limit***	***Description***
kImmediate	0x00	synchronous execution of restart (i.e. restart has happend when this method return).
kDefered	0x01	asynchronous execution of restart (i.e. restart is defered to later point in time)

Table 9.16: Implementation Data Type - ExecutionTypeType

9.1.1.16 RestartTypeType

Name	RestartTypeType
Kind	TYPE_REFERENCE
Derived from	`string`
Description	Represents the type of restart/reset.

Table 9.17: Implementation Data Type - RestartTypeType

9.1.2 Event memory management

9.1.2.1 MonitorActionType

Name	MonitorActionType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the status information reported by AAs being relevant for error monitoring.	
Range / Symbol	***Limit***	***Description***
kPassed	0x00	Monitor reports qualified test result passed.
kFailed	0x01	Monitor reports qualified test result failed
kPrepassed	0x02	Monitor reports unqualified test result pre-passed.

▽

△

kPrefailed	0x03	Monitor reports unqualified test result pre-failed.
kFdcThresholdReached	0x04	Monitor triggers the storage of ExtendedDataRecords and Freeze Frames (if the triggering condition is connected to this threshold).
kResetTestFailed	0x05	Reset TestFailed Bit without any other side effects like readiness
kFreezeDebouncing	0x06	Freeze the internal debounce counter/timer.
kResetDebouncing	0x07	Reset the internal debounce counter/timer.
kPrestore	0x08	Capture and prestores the freeze frame data.
kClearPrestore	0x09	Clears a prestored freeze frame.

Table 9.18: Implementation Data Type - MonitorActionType

9.1.2.2 DebouncingStateType

Name	DebouncingStateType			
Kind	TYPE_REFERENCE			
Derived from	`uint8_t`			
Description	DebouncingStateType			
Name	***Limit***	***Mask***	***State***	***Description***
kTemporarilyDefective	0	0x01	FALSE	Bit 0: Temporarily Defective (corresponds to 0 < FDC < 127)
kTemporarilyDefective	1	0x01	TRUE	Bit 0: Temporarily Defective (corresponds to 0 < FDC < 127)
kFinallyDefective	0	0x02	FALSE	Bit 1: finally Defective (corresponds to FDC = 127)
kFinallyDefective	2	0x02	TRUE	Bit 1: finally Defective (corresponds to FDC = 127)
kTemporarilyHealed	0	0x04	FALSE	Bit 2: temporarily healed (corresponds to -128 < FDC < 0)
kTemporarilyHealed	4	0x04	TRUE	Bit 2: temporarily healed (corresponds to -128 < FDC < 0)
kTestComplete	0	0x08	FALSE	Bit 3: Test complete (corresponds to FDC = -128 or FDC = 127)
kTestComplete	8	0x08	TRUE	Bit 3: Test complete (corresponds to FDC = -128 or FDC = 127)
kDTRUpdate	0	0x10	FALSE	Bit 4: DTR Update (= Test complete && Debouncing complete && enable conditions / storage conditions fulfilled)
kDTRUpdate	16	0x10	TRUE	Bit 4: DTR Update (= Test complete && Debouncing complete && enable conditions / storage conditions fulfilled)

Table 9.19: Implementation Data Type - DebouncingStateType

9.1.2.3 DTCFormatType

Name	DTCFormatType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	DTCFormatType	
Range / Symbol	*Limit*	*Description*
kDTCFormatOBD	0	
kDTCFormatUDS	1	
kDTCFormatJ1939	2	

Table 9.20: Implementation Data Type - DTCFormatType

9.1.2.4 DTCGroupType

Name	DTCGroupType
Kind	TYPE_REFERENCE
Derived from	`uint32_t`
Description	uint32

Table 9.21: Implementation Data Type - DTCGroupType

9.1.2.5 DTCStatusChangedType

Name	DTCStatusChangedType
Kind	STRUCTURE
Subelements	DTC `DTCType` udsDTCStatusByteOld `UdsDTCStatusByteType` udsDTCStatusByteNew `UdsDTCStatusByteType`
Derived from	-
Description	DTCStatusChangedType

Table 9.22: Implementation Data Type - DTCStatusChangedType

9.1.2.6 DTCType

Name	DTCType
Kind	TYPE_REFERENCE
Derived from	`uint32_t`
Description	uint32

Table 9.23: Implementation Data Type - DTCType

9.1.2.7 UdsDTCStatusByteType

Name	UdsDTCStatusByteType			
Kind	TYPE_REFERENCE			
Derived from	`uint8_t`			
Description	UdsDTCStatusByteType			
Name	*Limit*	*Mask*	*State*	*Description*
kTestFailed	0	0x01	FALSE	bit 0: TestFailed
kTestFailed	1	0x01	TRUE	bit 0: TestFailed
kTestFailedThisOperationCycle	0	0x02	FALSE	bit 1: TestFailedThisOperationCycle
kTestFailedThisOperationCycle	2	0x02	TRUE	bit 1: TestFailedThisOperationCycle
kPendingDTC	0	0x04	FALSE	bit 2: PendingDTC
kPendingDTC	4	0x04	TRUE	bit 2: PendingDTC
kConfirmedDTC	0	0x08	FALSE	bit 3: ConfirmedDTC
kConfirmedDTC	8	0x08	TRUE	bit 3: ConfirmedDTC
kTestNotCompletedSinceLast Clear	0	0x10	FALSE	bit 4: TestNotCompletedSinceLastClear
kTestNotCompletedSinceLast Clear	16	0x10	TRUE	bit 4: TestNotCompletedSinceLastClear
kTestFailedSinceLastClear	0	0x20	FALSE	bit 5: TestFailedSinceLastClear
kTestFailedSinceLastClear	32	0x20	TRUE	bit 5: TestFailedSinceLastClear
kTestNotCompletedThis OperationCycle	0	0x40	FALSE	bit 6: TestNotCompletedThisOperationCycle
kTestNotCompletedThis OperationCycle	64	0x40	TRUE	bit 6: TestNotCompletedThisOperationCycle
kWarningIndicatorRequested	0	0x80	FALSE	bit 7: WarningIndicatorRequested
kWarningIndicatorRequested	128	0x80	TRUE	bit 7: WarningIndicatorRequested

Table 9.24: Implementation Data Type - UdsDTCStatusByteType

9.1.2.8 EventStatusByteType

Name	EventStatusByteType			
Kind	TYPE_REFERENCE			
Derived from	`uint8_t`			
Description	EventStatusByteType			
Name	*Limit*	*Mask*	*State*	*Description*
kTestFailed	0	0x01	FALSE	bit 0: TestFailed
kTestFailed	1	0x01	TRUE	bit 0: TestFailed
kTestFailedThisOperationCycle	0	0x02	FALSE	bit 1: TestFailedThisOperationCycle
kTestFailedThisOperationCycle	2	0x02	TRUE	bit 1: TestFailedThisOperationCycle
kTestNotCompletedThis OperationCycle	0	0x40	FALSE	bit 6: TestNotCompletedThisOperationCycle
kTestNotCompletedThis OperationCycle	64	0x40	TRUE	bit 6: TestNotCompletedThisOperationCycle

Table 9.25: Implementation Data Type - EventStatusByteType

9.1.2.9 WIRStatusType

Name	WIRStatusType
Kind	TYPE_REFERENCE
Derived from	`bool`
Description	bool

Table 9.26: Implementation Data Type - WIRStatusType

9.1.2.10 FaultDetectionCounterType

Name	FaultDetectionCounterType
Kind	TYPE_REFERENCE
Derived from	`uint8_t`
Description	sint8

Table 9.27: Implementation Data Type - FaultDetectionCounterType

9.1.2.11 IndicatorStatusTyp

Name	IndicatorStatusType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	IndicatorStatusType	
Range / Symbol	***Limit***	***Description***
kOff	0x00	Indicator off mode
kContinous	0x01	Indicator continuously on mode
kBlinking	0x02	Indicator blinking mode
kBlinkingAndContinous	0x03	Indicator blinking or continuously on mode
kSlowFlash	0x04	Indicator slow flashing mode
kFastFlash	0x05	Indicator fast flashing mode
kOnDemand	0x06	Indicator on-demand mode
kShort	0x07	Indicator short mode

Table 9.28: Implementation Data Type - IndicatorStatusType

9.1.2.12 InitMonitorReasonType

Name	InitMonitorReasonType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	InitMonitorReasonType	
Range / Symbol	***Limit***	***Description***
kClear	0x00	Event was cleared and all internal values and states are reset.

▽

△

kRestart	0x01	Operation cycle of the event was (re-)started
kReenabled	0x02	Enable conditions or DTC settings re-enabled.
kStorageReenabled	0x03	Storage condition reenabled.

Table 9.29: Implementation Data Type - InitMonitorReasonType

9.1.2.13 OperationCycleStateType

Name	OperationCycleStateType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the state information of operation cycles.	
Range / Symbol	***Limit***	***Description***
kStart	0x00	Start/restart the operation cycle.
kEnd	0x01	End the operation cycle

Table 9.30: Implementation Data Type - OperationCycleStateType

9.1.2.14 SnapshotDataRecordType

Name	SnapshotDataRecordType
Kind	STRUCTURE
Subelements	snapshotRecordNumber `uint8_t` snapshotDataIdentifers `SnapshotDataIdentifersType`
Derived from	-
Description	Type containing a snapshot record number and its data of the snapshot record

Table 9.31: Implementation Data Type - SnapshotDataRecordType

9.1.2.15 SnapshotDataRecordsType

Name	SnapshotDataRecordsType
Kind	VECTOR
Subelements	`SnapshotDataRecordType`
Derived from	-
Description	SnapshotDataRecordsType

Table 9.32: Implementation Data Type - SnapshotDataRecordsType

9.1.2.16 SnapshotRecordUpdatedType

Name	SnapshotRecordUpdatedType
Kind	STRUCTURE
Subelements	DTC `DTCType` snapshotDataRecord `SnapshotDataRecordsType`
Derived from	-
Description	Contains the content of the updated snapshot record and the corresponding DTC number.

Table 9.33: Implementation Data Type - SnapshotRecordUpdatedType

9.1.2.17 SnapshotDataIdentifersType

Name	SnapshotDataIdentifersType
Kind	VECTOR
Subelements	`SnapshotDataIdentiferType`
Derived from	-
Description	SnapshotDataIdentifersType

Table 9.34: Implementation Data Type - SnapshotDataIdentifersType

9.1.2.18 SnapshotDataIdentiferType

Name	SnapshotDataIdentiferType
Kind	STRUCTURE
Subelements	dataIdentifer `uint16_t` dataElements `ByteVectorType`
Derived from	-
Description	Type containing a DID number and its data.

Table 9.35: Implementation Data Type - SnapshotDataIdentiferType

9.1.3 Diagnostic Over IP

9.1.3.1 GIDstatusType

Name	GIDstatusType
Kind	STRUCTURE
Subelements	GID `GIDType` furtherActionReq `uint8_t` syncStatus `uint8_t`
Derived from	-
Description	Type used in the method confirmed for the status of the service processing

Table 9.36: Implementation Data Type - GIDstatusType

9.1.3.2 GIDType

Name	GIDType
Kind	ARRAY
Subelements	`uint8_t`
Derived from	-
Description	6 byte array of uint8 representing the GID

Table 9.37: Implementation Data Type - GIDType

9.2 Provided Service Interfaces

This chapter lists all provided service interfaces of the DM.

9.2.1 Diagnostic service management

9.2.1.1 DiagnosticServer

Port

Name	DiagnosticServer_{SoftwareCluster}		
Kind	ProvidedPort	***Interface***	`DiagnosticServer`
Description	Provides information about the ControlDTCStatus		
Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`		

Table 9.38: Port - DiagnosticServer_{SoftwareCluster}

Service Interface

Name	DiagnosticServer
NameSpace	ara::diag

Table 9.39: Service Interfaces - DiagnosticServer

Fields

Name	ControlDTCStatus
Description	Contains the current status of the ControlDTCStatus
Type	`ControlDtcStatusType`
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	ENABLED

Table 9.40: Service Interface DiagnosticServer - Field: ControlDTCStatus

Methods

Name	EnableControlDTC
Description	Enforce restoring DTC setting.
FireAndForget	true

Table 9.41: Service Interface DiagnosticServer - Method: EnableControlDTC

9.2.1.2 DiagnosticConversation

Port

Name	DiagnosticConversation_{SoftwareCluster}_{Number}		
Kind	ProvidedPort	***Interface***	DiagnosticConversation
Description	Provides information about Diagnostic Conversation.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`  Number : MODEL.filterType("Number");`
	`WHERE Number.isWithin(SoftwareCluster);`		

Table 9.42: Port - DiagnosticConversation_{SoftwareCluster}_{Number}

Service Interface

Name	DiagnosticConversation
NameSpace	ara::diag

Table 9.43: Service Interfaces - DiagnosticConversation

Fields

Name	ActivityStatus
Description	Represents the status of an active conversation.
Type	ActivityStatusType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 9.44: Service Interface DiagnosticConversation - Field: ActivityStatus

Name	DiagnosticSession
Description	Represents the current active diagnostic session of an active conversation.
Type	DiagnosticSessionType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 9.45: Service Interface DiagnosticConversation - Field: DiagnosticSession

Name	DiagnosticSecurityLevel
Description	Represents the current active diagnostic security level of an active conversation.
Type	`DiagnosticSecurityLevelType`
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 9.46: Service Interface DiagnosticConversation - Field: DiagnosticSecurityLevel

Name	Identifier
Description	Contains the identifier (i.e. DiagnosticProtocolKind, TargetAddress, SourceAddress) of the diagnostic conversation.
Type	`DiagnosticConversationIdentifierType`
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 9.47: Service Interface DiagnosticConversation - Field: Identifier

Methods

Name	ResetToDefaultSession
Description	Method to reset the current session to default session.
FireAndForget	true

Table 9.48: Service Interface DiagnosticConversation - Method: ResetToDefaultSession

Name	Cancel
Description	Method to cancel the current diagnostic conversation. This includes current request execution and reset of any conversation-specific states i.e. Session or Security.
FireAndForget	true

Table 9.49: Service Interface DiagnosticConversation - Method: Cancel

9.2.2 Event memory management

9.2.2.1 DiagnosticEvent

Port

Name	DiagnosticEvent_{SoftwareCluster}_{DiagnosticEvent}		
Kind	ProvidedPort	***Interface***	DiagnosticEvent
Description	Provides information on diagnostic event.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticEvent : MODEL.filterType("DiagnosticEvent");`
	`WHERE DiagnosticEvent.isWithin(SoftwareCluster);`		

Table 9.50: Port - DiagnosticEvent_{SoftwareCluster}_{DiagnosticEvent}

Service Interface

Name	DiagnosticEvent
NameSpace	ara::diag

Table 9.51: Service Interfaces - DiagnosticEvent

Fields

Name	EventStatus
Description	Contains the current status of the event.
Type	EventStatusByteType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	0x40

Table 9.52: Service Interface DiagnosticEvent - Field: EventStatus

Name	WIRStatus
Description	Contains the current WIR status of the event.
Type	WIRStatusType
HasGetter	true
HasNotifier	true
HasSetter	true
Init-Value	false

Table 9.53: Service Interface DiagnosticEvent - Field: WIRStatus

Methods

Name	GetDTCNumber	
Description	Returns the DTC related to the event.	
Parameter	DTCFormat	
	Description	Define DTC format for the return value.
	Type	`DTCFormatType`
	Variation	
	Direction	IN
Parameter	DTCOfEvent	
	Description	DTC number in respective DTCFormatType
	Type	`DTCType`
	Variation	
	Direction	OUT
Application Error Set	`Common`	This error set include the error to reject the restart execution.

Table 9.54: Service Interface DiagnosticEvent - Method: GetDTCNumber

Name	GetDebouncingStatus	
Description	Gets the debouncing status of an event.	
Parameter	debouncingState	
	Description	Current debouncing state
	Type	`DebouncingStateType`
	Variation	
	Direction	OUT

Table 9.55: Service Interface DiagnosticEvent - Method: GetDebouncingStatus

Name	GetFaultDetectionCounter	
Description	Returns the current value of fault detection counter of this event.	
Parameter	faultDetectionCounter	
	Description	Current FaultDetectionCounter value
	Type	`FaultDetectionCounterType`
	Variation	
	Direction	OUT

Table 9.56: Service Interface DiagnosticEvent - Method: GetFaultDetectionCounter

9.2.2.2 DTCInformation

Port

Name	DTCInformation_{SoftwareCluster}_{DiagnosticMemoryDestination}		
Kind	ProvidedPort	***Interface***	DTCInformation
Description	Provides information on diagnostic event.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticMemoryDestination : MODEL.filterType("DiagnosticMemoryDestination");`
	`WHERE DiagnosticMemoryDestination.isWithin(SoftwareCluster);`		

Table 9.57: Port - DTCInformation_{SoftwareCluster}_{DiagnosticMemoryDestination}

Service Interface

Name	DTCInformation
NameSpace	ara::diag

Table 9.58: Service Interfaces - DTCInformation

Methods

Name	GetCurrentStatus	
Description	Determines the status of a DTC.	
Parameter	DTC	
	Description	DTC whose status is requested.
	Type	DTCType
	Variation	
	Direction	IN
Parameter	UDSDTCStatusByte	
	Description	Contains the status of the DTC.
	Type	UdsDTCStatusByteType
	Variation	
	Direction	OUT
Application Error Set	Common	This error set include the error to reject the restart execution.

Table 9.59: Service Interface DTCInformation - Method: GetCurrentStatus

Events

Name	DTCStatusChanged
Description	Notification about the change in the status of a DTC.
Type	DTCStatusChangedType

Table 9.60: Service Interface DTCInformation - Event: DTCStatusChanged

Name	SnapshotRecordUpdated
Description	Notification about an update of a SnapshotRecord.
Type	`SnapshotRecordUpdatedType`

Table 9.61: Service Interface DTCInformation - Event: SnapshotRecordUpdated

9.2.2.3 DiagnosticMemory

Port

Name	DiagnosticMemory_{SoftwareCluster}_{DiagnosticMemoryDestination}		
Kind	ProvidedPort	***Interface***	DiagnosticMemory
Description	Provides access to the fault memories.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticMemoryDestination : MODEL.filterType("DiagnosticMemoryDestination");`
	`WHERE DiagnosticMemoryDestination.isWithin(SoftwareCluster);`		

Table 9.62: Port - DiagnosticMemory_{SoftwareCluster}_{DiagnosticMemoryDestination}

Service Interface

Name	DiagnosticMemory
NameSpace	ara::diag

Table 9.63: Service Interfaces - DiagnosticMemory

Fields

Name	NumberOfStoredEntries
Description	Numbner of currently stored fault memory entries.
Type	uint32_t
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	0

Table 9.64: Service Interface DiagnosticMemory - Field: NumberOfStoredEntries

Methods

Name	Clear	
Description	Method for Clearing a DTC or a group of DTCs.	
Parameter	DTC	
	Description	DTC group to be cleared.
	Type	DTCGroupType
	Variation	
	Direction	IN
Application Error Set	ClearFailedReason	This error set includes the different kind of rejections to clear DTCs.

Table 9.65: Service Interface DiagnosticMemory - Method: Clear

9.2.2.4 EnableCondition

Port

Name	EnableCondition_{SoftwareCluster}_{DiagnosticEnableCondition}		
Kind	ProvidedPort	***Interface***	EnableCondition
Description	Provides functionality for handling of enable conditions.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticEnableCondition : MODEL.filterType("DiagnosticEnableCondition");`
	`WHERE DiagnosticEnableCondition.isWithin(SoftwareCluster);`		

Table 9.66: Port - EnableCondition_{SoftwareCluster}_{DiagnosticEnableCondition}

Service Interface

Name	EnableCondition
NameSpace	ara::diag

Table 9.67: Service Interfaces - EnableCondition

Fields

Name	State
Description	Contains the current state of an enable condition.
Type	StateType
HasGetter	true
HasNotifier	false
HasSetter	true
Init-Value	0

Table 9.68: Service Interface EnableCondition - Field: State

9.2.2.5 StorageCondition

Port

Name	StorageCondition_{SoftwareCluster}_{DiagnosticStorageCondition}		
Kind	ProvidedPort	***Interface***	StorageCondition
Description	Provides functionality for handling of storage conditions.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticStorageCondition : MODEL.filterType("DiagnosticStorageCondition");`
	`WHERE DiagnosticStorageCondition.isWithin(SoftwareCluster);`		

Table 9.69: Port - StorageCondition_{SoftwareCluster}_{DiagnosticStorageCondition}

Service Interface

Name	StorageCondition
NameSpace	ara::diag

Table 9.70: Service Interfaces - StorageCondition

Fields

Name	State
Description	Contains the current state of an storage condition.
Type	StateType
HasGetter	true
HasNotifier	false
HasSetter	true
Init-Value	To be done: specify value

Table 9.71: Service Interface StorageCondition - Field: State

9.2.2.6 ClearCondition

Port

Name	ClearCondition_{SoftwareCluster}_{DiagnosticClearCondition}		
Kind	ProvidedPort	***Interface***	ClearCondition
Description	Provides functionality for handling of clear conditions.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticClearCondition : MODEL.filterType("DiagnosticClearCondition");`
	`WHERE DiagnosticClearCondition.isWithin(SoftwareCluster);`		

Table 9.72: Port - ClearCondition_{SoftwareCluster}_{DiagnosticClearCondition}

Service Interface

Name	ClearCondition
NameSpace	ara::diag

Table 9.73: Service Interfaces - ClearCondition

Fields

Name	State
Description	Contains the current state of an clear DTC condition.
Type	StateType
HasGetter	true
HasNotifier	false
HasSetter	true
Init-Value	0

Table 9.74: Service Interface ClearCondition - Field: State

9.2.2.7 OperationCycle

Port

Name	OperationCycle_{SoftwareCluster}_{DiagnosticOperationCycle}		
Kind	ProvidedPort	***Interface***	OperationCycle
Description	Provides functionality for handling of operation cycles.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticOperationCycle : MODEL.filterType("DiagnosticOperationCycle");`
	`WHERE DiagnosticOperationCycle.isWithin(SoftwareCluster);`		

Table 9.75: Port - OperationCycle_{SoftwareCluster}_{DiagnosticOperationCycle}

Service Interface

Name	OperationCycle
NameSpace	ara::diag

Table 9.76: Service Interfaces - OperationCycle

Fields

Name	State
Description	Contains the current state of an operation cycle.
Type	OperationCycleStateType
HasGetter	true
HasNotifier	true
HasSetter	true
Init-Value	To be done: specify value

Table 9.77: Service Interface OperationCycle - Field: State

9.2.2.8 Indicator

Port

Name	Indicator_{SoftwareCluster}_{DiagnosticIndicator}		
Kind	ProvidedPort	***Interface***	Indicator
Description	Provides functionality for handling of indicators.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticIndicator : MODEL.filterType("DiagnosticIndicator");`
	`WHERE DiagnosticIndicator.isWithin(SoftwareCluster);`		

Table 9.78: Port - Indicator_{SoftwareCluster}_{DiagnosticIndicator}

Service Interface

Name	Indicator
NameSpace	ara::diag

Table 9.79: Service Interfaces - Indicator

Fields

Name	IndicatorStatus
Description	Contains the current state of an indicator.
Type	IndicatorStatusType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	IndicatorStatusType::kOff

Table 9.80: Service Interface Indicator - Field: IndicatorStatus

9.3 Required Service Interfaces

This chapter lists all required service interfaces of the DM.

9.3.1 Diagnostic service management

9.3.1.1 GenericUDSService

Port

Name	GenericUDSService_{SoftwareCluster}_{DiagnosticServiceSwMapping}		
Kind	RequiredPort	***Interface***	GenericUDSService
Description	Generic handler for UDS services. Can be mapped to any single diagnostic object (SID, Sub-Function, DID, RID,...).		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticServiceSwMapping : MODEL.filterType("DiagnosticServiceSwMapping");`
	`WHERE DiagnosticServiceSwMapping.isWithin(SoftwareCluster);`		

Table 9.81: Port - GenericUDSService_{SoftwareCluster}_{DiagnosticServiceSwMapping}

Service Interface

Name	GenericUDSService
NameSpace	ara::diag

Table 9.82: Service Interfaces - GenericUDSService

Methods

Name	HandleMessage	
Description	Called for any request messsage of the mapped DiagnosticServiceSwMapping.	
Parameter	SID	
	Description	Diagnostic Request Service Identifier.
	Type	SIDType
	Variation	
	Direction	IN
Parameter	requestData	
	Description	Diagnostic request data (starting after SID).
	Type	ByteVectorType
	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	responseData	
	Description	Diagnostic response data (starting after SID).
	Type	ByteVectorType
	Variation	
	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.83: Service Interface GenericUDSService - Method: HandleMessage

Name	Cancel	
Description	Called if the current conversation is canceled.	
FireAndForget	true	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	
	Direction	IN

Table 9.84: Service Interface GenericUDSService - Method: Cancel

9.3.1.2 ServiceManufacturerValidation

Port

Name	ServiceManufacturerValidation_{SoftwareCluster}		
Kind	RequiredPort	***Interface***	ServiceManufacturerValidation
Description	This service is the manufacturer notification handler.		
Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`		

Table 9.85: Port - ServiceManufacturerValidation_{SoftwareCluster}

Service Interface

Name	ServiceManufacturerValidation
NameSpace	ara::diag

Table 9.86: Service Interfaces - ServiceManufacturerValidation

Methods

Name	Validate	
Description	Called for any request message.	
Parameter	requestData	
	Description	Diagnostic request message including SID.
	Type	ByteVectorType
	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Application Error Set	UDSNegativeResponseCodeValidation	This error set includes all NegativeResponseCodes defined in UDS + the possbility to reject the request message without any response message.

Table 9.87: Service Interface ServiceManufacturerValidation - Method: Validate

Name	Confirmation	
Description	This method is called by DM on a configured manufacturer/supplier notification handler, when a diagnostic request has been finished, to notify the handler about the outcome.	
FireAndForget	true	
Parameter	status	
	Description	status/outcome of the service processing.
	Type	ConfirmationStatusType

▽

△

	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	
	Direction	IN

Table 9.88: Service Interface ServiceManufacturerValidation - Method: Confirmation

9.3.1.3 ServiceSupplierValidation

Port

Name	ServiceSupplierValidation_{SoftwareCluster}		
Kind	RequiredPort	***Interface***	ServiceSupplierValidation
Description	This service is the supplier notification handler.		
Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`		

Table 9.89: Port - ServiceSupplierValidation_{SoftwareCluster}

Service Interface

Name	ServiceSupplierValidation
NameSpace	ara::diag

Table 9.90: Service Interfaces - ServiceSupplierValidation

Methods

Name	Validate	
Description	Called for any request message.	
Parameter	requestData	
	Description	Diagnostic request message including SID.
	Type	ByteVectorType
	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Application Error Set	UDSNegativeResponseCodeValidation	This error set includes all NegativeResponseCodes defined in UDS + the possbility to reject the request message without any response message.

Table 9.91: Service Interface ServiceSupplierValidation - Method: Validate

Name	Confirmation	
Description	This method is called by DM on a configured manufacturer/supplier notification handler, when a diagnostic request has been finished, to notify the handler about the outcome.	
FireAndForget	true	
Parameter	status	
	Description	status/outcome of the service processing.
	Type	ConfirmationStatusType

▽

△

	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	
	Direction	IN

Table 9.92: Service Interface ServiceSupplierValidation - Method: Confirmation

9.3.1.4 RoutineService

Port

Name	RoutineService_{SoftwareCluster}_{DiagnosticRoutine}		
Kind	RequiredPort	***Interface***	RoutineService
Description	Requires application providing methods of configured signature.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticRoutine : MODEL.filterType("DiagnosticRoutine");`
	`WHERE DiagnosticRoutine.isWithin(SoftwareCluster);`		

Table 9.93: Port - RoutineService_{SoftwareCluster}_{DiagnosticRoutine}

Service Interface

Name	RoutineService_{SoftwareCluster}_{DiagnosticRoutine}
NameSpace	ara::diag

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticRoutine : MODEL.filterType("DiagnosticRoutine");`
	`WHERE DiagnosticRoutine.isWithin(SoftwareCluster);`

Table 9.94: Service Interfaces - RoutineService

Methods

Name	Start	
Description	Called for sub-function start of a routine.	
Parameter	req_{DiagnosticDataElement}	
	Description	IN-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	custom

	Variation	`FOR arg : Method.getAttribute("request");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	custom

	Variation	`FOR arg : Method.getAttribute("response");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.95: Service Interface RoutineService - Method: Start

Name	Stop	
Description	Called for sub-function stop of a routine if configured.	
Parameter	req_{DiagnosticDataElement}	
	Description	IN-Parameter of the stop sub-function according to DiagnosticRoutine.
	Type	custom

	Variation	`FOR arg : Method.getAttribute("request");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	custom

	Variation	`FOR arg : Method.getAttribute("response");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.96: Service Interface RoutineService - Method: Stop

Name	RequestResults	
Description	Called for sub-function requestRoutineResults of a routine if configured.	
Parameter	req_{DiagnosticDataElement}	
	Description	IN-Parameter of the requestResults sub-function according to DiagnosticRoutine.
	Type	custom

	Variation	`FOR arg : Method.getAttribute("request");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the requestRoutineResults sub-function according to Diagnostic Routine.
	Type	custom

▽

△

	Variation	`FOR arg : Method.getAttribute("response");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.97: Service Interface RoutineService - Method: RequestResults

Name	Cancel	
Description	Called if the current conversation is canceled.	
FireAndForget	true	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN

Table 9.98: Service Interface RoutineService - Method: Cancel

9.3.1.5 SecurityAccess

Port

Name	SecurityAccess_{SoftwareCluster}_{DiagnosticSecurityLevel}		
Kind	RequiredPort	***Interface***	SecurityAccess
Description	SecurityAccess.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticSecurityLevel : MODEL.filterType("DiagnosticSecurityLevel");`
	`WHERE DiagnosticSecurityLevel.isWithin(SoftwareCluster);`		

Table 9.99: Port - SecurityAccess_{SoftwareCluster}_{DiagnosticSecurityLevel}

Service Interface

Name	SecurityAccess
NameSpace	ara::diag

Table 9.100: Service Interfaces - SecurityAccess

Methods

Name	GetSeed	
Description	Called for SecurityAccess (x027) with subfunction requestSeedId if configured (see DiagnosticSecurity Access)	
Parameter	securityAccessDataRecord	
	Description	provided securityAccessDataRecord
	Type	ByteVectorType
	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	seed	
	Description	provided seed
	Type	ByteVectorType
	Variation	
	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.101: Service Interface SecurityAccess - Method: GetSeed

Name	CompareKey	
Description	Called for SecurityAccess (x027) with subfunction sendKey if configured (see DiagnosticSecurityAccess).	
Parameter	key	
	Description	The key to be validated
	Type	ByteVectorType
	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	result	
	Description	Result of the key validation.
	Type	KeyCompareResultType
	Variation	
	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.102: Service Interface SecurityAccess - Method: CompareKey

Name	Cancel	
Description	Called if the current conversation is canceled.	
FireAndForget	true	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN

Table 9.103: Service Interface SecurityAccess - Method: Cancel

9.3.1.6 CommunicationControl

Port

Name	CommunicationControl_{SoftwareCluster}		
Kind	RequiredPort	***Interface***	CommunicationControl
Description	CommunicationControl.		
Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`		

Table 9.104: Port - CommunicationControl_{SoftwareCluster}

Service Interface

Name	CommunicationControl
NameSpace	ara::diag

Table 9.105: Service Interfaces - CommunicationControl

Methods

Name	CommCtrlRequest	
Description	Called for CommunicationControl (x028) with any subfunction as subfunction value is part of argument list. Typically provider of this interface is considered as part of the state management.	
Parameter	controlType	
	Description	All UDS request parameters packed into a structure since it holds optional elementr
	Type	ComCtrlRequestParamsType
	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.106: Service Interface CommunicationControl - Method: CommCtrlRequest

9.3.1.7 RequestRestart

Port

Name	RequestRestart_{SoftwareCluster}		
Kind	RequiredPort	***Interface***	RequestRestart
Description	This is the service interface for requesting a restart of the machine, SWCL or application.		
Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`		

Table 9.107: Port - RequestRestart_{SoftwareCluster}

Service Interface

Name	RequestRestart
NameSpace	ara::diag

Table 9.108: Service Interfaces - RequestRestart

Methods

Name	RequestRestart	
Description	Requests a restart of SoftwareCluster(s) or parts of SoftwareCluster(s) running on the machine.	
Parameter	RestartType	
	Description	Type of the requested reset.
	Type	RestartTypeType
	Variation	
	Direction	IN
Parameter	ExecutionType	
	Description	Type of execution: Immediate is synchronous (i.e. restart has happend when this method return); Defered is asynchronous (i.e. restart is defered to later point in time)
	Type	ExecutionTypeType
	Variation	
	Direction	IN
Application Error Set	RequestRestart	This error set include the error to reject the restart execution.

Table 9.109: Service Interface RequestRestart - Method: RequestRestart

9.3.2 Event memory management

9.3.2.1 DiagnosticMonitor

Port

Name	DiagnosticMonitor_{SoftwareCluster}_{DiagnosticEvent}		
Kind	RequiredPort	***Interface***	DiagnosticMonitor
Description	Requires diagnostic monitor reporting results of diagnostic event monitoring.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticEvent : MODEL.filterType("DiagnosticEvent");`
	`WHERE DiagnosticEvent.isWithin(SoftwareCluster);`		

Table 9.110: Port - DiagnosticMonitor_{SoftwareCluster}_{DiagnosticEvent}

Service Interface

Name	DiagnosticMonitor
NameSpace	ara::diag

Table 9.111: Service Interfaces - DiagnosticMonitor

Methods

Name	InitMonitor	
Description	Event-specific notification for monitors about clearing, operation cycle restart or enable/storage condition re-enabling.	
FireAndForget	true	
Parameter	reason	
	Description	The reason for initiallizing the monitor.
	Type	InitMonitorReasonType
	Variation	
	Direction	IN

Table 9.112: Service Interface DiagnosticMonitor - Method: InitMonitor

Events

Name	MonitorAction
Description	Contains either the last (un-)qualified test result of the diagnostic monitor or commands to control the debouncing or to force a prestorage.
Type	MonitorActionType

Table 9.113: Service Interface DiagnosticMonitor - Event: MonitorAction

9.3.3 DoIP protocol

9.3.3.1 DoIPGroupIdentification

Port

Name	DoIPGroupIdentification		
Kind	RequiredPort	***Interface***	DoIPGroupIdentification
Description	DoIPGroupIdentification		
Variation			

Table 9.114: Port - DoIPGroupIdentification

Service Interface

Name	DoIPGroupIdentification
NameSpace	ara::diag

Table 9.115: Service Interfaces - DoIPGroupIdentification

Fields

Name	GIDstatus
Description	Contains the current GID state for the DoIP protocol.
Type	GIDstatusType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 9.116: Service Interface DoIPGroupIdentification - Field: GIDstatus

9.3.3.2 DoIPPowerModeInformation

Port

Name	DoIPPowerModeInformation		
Kind	RequiredPort	***Interface***	DoIPPowerModeInformation
Description	DoIPPowerModeInformation		
Variation			

Table 9.117: Port - DoIPPowerModeInformation

Service Interface

Name	DoIPPowerModeInformation
NameSpace	ara::diag

Table 9.118: Service Interfaces - DoIPPowerModeInformation

Fields

Name	PowerMode
Description	Contains the current power state for the DoIP protocol.
Type	`uint8_t`
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 9.119: Service Interface DoIPPowerModeInformation - Field: PowerMode

9.3.4 Common

9.3.4.1 DataElement

Port

Name	DataElement_{SoftwareCluster}_{DiagnosticDataElement}		
Kind	RequiredPort	***Interface***	DataElement
Description	This is the service interface for any DiagnosticDataElement which is not used within a DataIdentifier service.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticDataElement : MODEL.filterType("DiagnosticDataElement");`
	`WHERE DiagnosticDataElement.isWithin(SoftwareCluster);`		

Table 9.120: Port - DataElement_{SoftwareCluster}_{DiagnosticDataElement}

Service Interface

Name	DataElement_{SoftwareCluster}_{DiagnosticDataElement}
NameSpace	ara::diag::DataElement

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticDataElement : MODEL.filterType("DiagnosticDataElement");`
	`WHERE DiagnosticDataElement.isWithin(SoftwareCluster);`

Table 9.121: Service Interfaces - DataElement

Methods

Name	Read	
Description	Called for data aquisition of a DiagnosticDataElement.	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	OUT-Parameter.
	Type	custom
	Variation	
	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.122: Service Interface DataElement - Method: Read

Name	Cancel	
Description	Called if the current conversation is canceled.	
FireAndForget	true	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	
	Direction	IN

Table 9.123: Service Interface DataElement - Method: Cancel

9.3.4.2 DataIdentifier

Port

Name	DataIdentifier_{SoftwareCluster}_{DiagnosticDataIdentifier}		
Kind	RequiredPort	***Interface***	DataIdentifier
Description	This is the default service interface for a DiagnosticDataIdentifier.		

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticDataIdentifier : MODEL.filterType("DiagnosticDataIdentifier");`
	`WHERE DiagnosticDataIdentifier.isWithin(SoftwareCluster);`		

Table 9.124: Port - DataIdentifier_{SoftwareCluster}_{DiagnosticDataIdentifier}

Service Interface

Name	DataIdentifier_{SoftwareCluster}_{DiagnosticDataIdentifier}
NameSpace	ara::diag

Variation	`FOR SoftwareCluster : MODEL.filterType("SoftwareCluster");`
	`DiagnosticDataIdentifier : MODEL.filterType("DiagnosticDataIdentifier");`
	`WHERE DiagnosticDataIdentifier.isWithin(SoftwareCluster);`

Table 9.125: Service Interfaces - DataIdentifier

Methods

Name	Read	
Description	Called for ReadDataByIdentifer request for this DiagnosticDataIdentifier (if configured).	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	DataElement within response message.
	Type	custom

	Variation	`FOR arg : Method.getAttribute("dataElement");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	OUT
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.126: Service Interface DataIdentifier - Method: Read

Name	Write	
Description	Called for WriteDataByIdentifer request for this DiagnosticDataIdentifier (if configured).	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	DataElement within the request message.
	Type	custom

	Variation	`FOR arg : Method.getAttribute("dataElement");`
		`LET DiagnosticDataElement = arg.getAttribute("dataElement");`
		`swDataDefProps = DiagnosticDataElement.getAttribute("swDataDefProps");`
		`BaseTypeRef = swDataDefProps.getAttribute("baseTypeRef");`

	Direction	IN
Application Error Set	UDSNegativeResponseCode	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.127: Service Interface DataIdentifier - Method: Write

Name	Cancel	
Description	Called if the current conversation is canceled.	
FireAndForget	true	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType
	Variation	
	Direction	IN

Table 9.128: Service Interface DataIdentifier - Method: Cancel

9.3.4.3 VINInformation

Port

Name	VINInformation		
Kind	RequiredPort	***Interface***	VINInformation
Description	Requires handler for VIN information.		
Variation			

Table 9.129: Port - VINInformation

Service Interface

Name	VINInformation
NameSpace	ara::diag

Table 9.130: Service Interfaces - VINInformation

Methods

Name	Read	
Description	Called for reading the VIN.	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	
	Direction	IN
Parameter	vin	
	Description	VIN data.
	Type	`VINType`
	Variation	
	Direction	OUT
Application Error Set	`UDSNegativeResponseCode`	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.131: Service Interface VINInformation - Method: Read

Name	Write	
Description	Called for writing the VIN.	
Parameter	vin	
	Description	VIN data.
	Type	`VINType`
	Variation	
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	
	Direction	IN
Application Error Set	`UDSNegativeResponseCode`	This error set includes all NegativeResponseCodes defined in UDS.

Table 9.132: Service Interface VINInformation - Method: Write

Name	Cancel	
Description	Called if the current operation is canceled.	
FireAndForget	true	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	
	Direction	IN

Table 9.133: Service Interface VINInformation - Method: Cancel

9.4 Application Errors

This chapter lists all application errors of the DM.

9.4.1 Application Error Domain

9.4.1.1 Diag_Common

Name	***Code***	***Description***
kNoSuchDTC	0	No DTC available.

Table 9.134: Application Errors of Diag_Common

9.4.1.2 Diag_RequestRestart

Name	***Code***	***Description***
kRejected	0	Reject the request to restart.

Table 9.135: Application Errors of Diag_RequestRestart

9.4.1.3 Diag_ClearFailedReason

Name	***Code***	***Description***
kFailed	0	Failed to clear the DTC due to any other reason.
kBusy	1	DTC not cleared, as another clearing process is in progress. The caller can retry later.
kMemoryError	2	An error occurred during erasing a memory location.
kWrongDtc	3	DTC value does not exist in the current configuration

Table 9.136: Application Errors of Diag_ClearFailedReason

9.4.1.4 Diag_UDSNegativeResponseCode

Name	*Code*	*Description*
kGeneralReject	0x10	According to ISO.
kServiceNotSupported	0x11	According to ISO.
kSubfunctionNotSupported	0x12	According to ISO.
kIncorrectMessageLengthOrInvalidFormat	0x13	According to ISO.
kResponseTooLong	0x14	According to ISO.
kBusyRepeatRequest	0x21	According to ISO.
kConditionsNotCorrect	0x22	According to ISO.
kRequestSequenceError	0x24	According to ISO.
kNoResponseFromSubnetComponent	0x25	According to ISO.
kFailurePreventsExecutionOfRequested Action	0x26	According to ISO.
kRequestOutOfRange	0x31	According to ISO.
kSecurityAccessDenied	0x33	According to ISO.
kInvalidKey	0x35	According to ISO.
kExceedNumberOfAttempts	0x36	According to ISO.
kRequiredTimeDelayNotExpired	0x37	According to ISO.
kUploadDownloadNotAccepted	0x70	According to ISO.
kTransferDataSuspended	0x71	According to ISO.
kGeneralProgrammingFailure	0x72	According to ISO.
kWrongBlockSequenceCounter	0x73	According to ISO.
kSubFunctionNotSupportedInActive Session	0x7E	According to ISO.
kServiceNotSupportedInActiveSession	0x7F	According to ISO.
kRpmTooHigh	0x81	According to ISO.
kRpmTooLow	0x82	According to ISO.
kEngineIsRunning	0x83	According to ISO.
kEngineIsNotRunning	0x84	According to ISO.
kEngineRunTimeTooLow	0x85	According to ISO.
kTemperatureTooHigh	0x86	According to ISO.
kTemperatureTooLow	0x87	According to ISO.
kVehicleSpeedTooHigh	0x88	According to ISO.
kVehicleSpeedTooLow	0x89	According to ISO.
kThrottlePedalTooHigh	0x8A	According to ISO.
kThrottlePedalTooLow	0x8B	According to ISO.
kTransmissionRangeNotInNeutral	0x8C	According to ISO.
kTransmissionRangeNotInGear	0x8D	According to ISO.
kBrakeSwitchNotClosed	0x8F	According to ISO.
kShifterLeverNotInPark	0x90	According to ISO.
kTorqueConverterClutchLocked	0x91	According to ISO.
kVoltageTooHigh	0x92	According to ISO.
kVoltageTooLow	0x93	According to ISO.
kNoProcessingNoResponse	0xFF	Deviating from ISO - no further service processing and no response (silently ignore request message).

Table 9.137: Application Errors of Diag_UDSNegativeResponseCode

9.4.2 Application Error Sets

9.4.2.1 Diag_Common

Error Set Name	Common
Description	This error set include the error to reject the restart execution.
Reference	kNoSuchDTC

Table 9.138: Application Errors of Set Common

9.4.2.2 Diag_RequestRestart

Error Set Name	RequestRestart
Description	This error set include the error to reject the restart execution.
Reference	kRejected

Table 9.139: Application Errors of Set RequestRestart

9.4.2.3 Diag_ClearFailedReason

Error Set Name	ClearFailedReason
Description	This error set includes the different kind of rejections to clear DTCs.
Reference	kBusy, kFailed, kMemoryError, kWrongDtc

Table 9.140: Application Errors of Set ClearFailedReason

9.4.2.4 Diag_UDSNegativeResponseCode

Error Set Name	UDSNegativeResponseCode
Description	This error set includes all NegativeResponseCodes defined in UDS.
Reference	kBrakeSwitchNotClosed, kBusyRepeatRequest, kConditionsNotCorrect, kEngineIsNotRunning, kEngineIsRunning, kEngineRunTimeTooLow, kExceedNumberOfAttempts, kFailurePreventsExecutionOfRequestedAction, kGeneralReject, kGeneralProgrammingFailure, kIncorrectMessageLengthOrInvalidFormat, kInvalidKey, kNoResponseFromSubnetComponent, kResponseTooLong, kRequestOutOfRange, kRequestSequenceError, kRequiredTimeDelayNotExpired, kRpmTooHigh, kRpmTooLow, kSecurityAccessDenied, kServiceNotSupported, kServiceNotSupportedInActiveSession, kShifterLeverNotInPark, kSubfunctionNotSupported, kSubFunctionNotSupportedInActiveSession, kTemperatureTooHigh, kTemperatureTooLow, kTorqueConverterClutchLocked, kThrottlePedalTooHigh, kThrottlePedalTooLow, kTransferDataSuspended, kTransmissionRangeNotInGear, kTransmissionRangeNotInNeutral, kUploadDownloadNotAccepted, kVehicleSpeedTooHigh, kVehicleSpeedTooLow, kVoltageTooHigh, kVoltageTooLow, kWrongBlockSequenceCounter

Table 9.141: Application Errors of Set UDSNegativeResponseCode

9.4.2.5 Diag_UDSNegativeResponseCodeValidation

Error Set Name	UDSNegativeResponseCodeValidation
Description	This error set includes all NegativeResponseCodes defined in UDS + the possbility to reject the request message without any response message.
Reference	kBrakeSwitchNotClosed, kBusyRepeatRequest, kConditionsNotCorrect, kEngineIsNotRunning, kEngineIsRunning, kEngineRunTimeTooLow, kExceedNumberOfAttempts, kFailurePreventsExecutionOfRequestedAction, kGeneralReject, kGeneralProgrammingFailure, kIncorrectMessageLengthOrInvalidFormat, kInvalidKey, kNoProcessingNoResponse, kNoResponseFromSubnetComponent, kResponseTooLong, kRequestOutOfRange, kRequestSequenceError, kRequiredTimeDelayNotExpired, kRpmTooHigh, kRpmTooLow, kSecurityAccessDenied, kServiceNotSupported, kServiceNotSupportedInActiveSession, kShifterLeverNotInPark, kSubfunctionNotSupported, kSubFunctionNotSupportedInActiveSession, kTemperatureTooHigh, kTemperatureTooLow, kTorqueConverterClutchLocked, kThrottlePedalTooHigh, kThrottlePedalTooLow, kTransferDataSuspended, kTransmissionRangeNotInGear, kTransmissionRangeNotInNeutral, kUploadDownloadNotAccepted, kVehicleSpeedTooHigh, kVehicleSpeedTooLow, kVoltageTooHigh, kVoltageTooLow, kWrongBlockSequenceCounter

Table 9.142: Application Errors of Set UDSNegativeResponseCodeValidation

A Mentioned Manifest Elements

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

Class	***DataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of any data type.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level.

Table A.1: DataPrototype

Class	**DiagEventDebounceCounterBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the counter-based debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemDebounceAlgorithmClass to DemDebounce CounterBased.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
counterBased FdcThreshold StorageValue	Integer	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.

▽

△

Class	**DiagEventDebounceCounterBased**			
counter DecrementStep Size	Integer	1	attr	This value shall be taken to decrement the internal debounce counter.
counterFailed Threshold	Integer	1	attr	This value defines the event-specific limit that indicates the "failed" counter status.
counter IncrementStep Size	Integer	1	attr	This value shall be taken to increment the internal debounce counter.
counterJump Down	Boolean	1	attr	This value activates or deactivates the counter jump-down behavior.
counterJump DownValue	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from incrementing to decrementing.
counterJumpUp	Boolean	1	attr	This value activates or deactivates the counter jump-up behavior.
counterJumpUp Value	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from decrementing to incrementing.
counterPassed Threshold	Integer	1	attr	This value defines the event-specific limit that indicates the "passed" counter status.

Table A.2: DiagEventDebounceCounterBased

Class	**DiagEventDebounceTimeBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the time-based pre-debounce algorithm shall be used by the Dem for this diagnostic monitor. This is related to set the EcuC choice container DemDebounceAlgorithmClass to DemDebounceTime Base.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
timeBasedFdc Threshold StorageValue	TimeValue	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.
timeFailed Threshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "failed" status.
timePassed Threshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "passed" status.

Table A.3: DiagEventDebounceTimeBased

Class	***DiagnosticAbstractDataIdentifier*** (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class represents an abstract base class for the modeling of a diagnostic data identifier (DID).			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticDataIdentifier, DiagnosticDynamicDataIdentifier			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
id	PositiveInteger	1	attr	This is the numerical identifier used to identify the DiagnosticAbstractDataIdentifier in the scope of diagnostic workflow

Table A.4: DiagnosticAbstractDataIdentifier

Class	**DiagnosticAccessPermission**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm			
Note	This represents the specification of whether a given service can be accessed according to the existence of meta-classes referenced by a particular DiagnosticAccessPermission. In other words, this meta-class acts as a mapping element between several (otherwise unrelated) pieces of information that are put into context for the purpose of checking for access rights. **Tags:** atp.recommendedPackage=DiagnosticAccessPermissions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnostic Session	DiagnosticSession	*	ref	This represents the associated DiagnosticSessions
environmental Condition	Diagnostic EnvironmentalCondition	0..1	ref	This represents the environmental conditions associated with the access permission.
securityLevel	DiagnosticSecurityLevel	*	ref	This represents the associated DiagnosticSecurityLevels

Table A.5: DiagnosticAccessPermission

Class	**DiagnosticAging**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticAging			
Note	Defines the aging algorithm. **Tags:** atp.recommendedPackage=DiagnosticAgings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
agingCycle	DiagnosticOperation Cycle	0..1	ref	This represents the applicable aging cycle. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=agingCycle, variationPoint.ShortLabel vh.latestBindingTime=preCompileTime
threshold	PositiveInteger	0..1	attr	Number of aging cycles needed to unlearn/delete the event. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table A.6: DiagnosticAging

Class	**DiagnosticClearCondition**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::DiagnosticDesign::DiagnosticClearCondition			
Note	This meta-class describes a clear condition for diagnostic purposes. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticConditions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticCondition*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.7: DiagnosticClearCondition

Enumeration	**DiagnosticClearDtcLimitationEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticCommonProps
Note	Scope of the DEM_ClearDTC Api.
Literal	***Description***
allSupportedDtcs	DEM_ClearDtc API accepts all supported DTC values. **Tags:** atp.EnumerationValue=0
clearAllDtcs	DEM_ClearDtc API accepts ClearAllDTCs only. **Tags:** atp.EnumerationValue=1

Table A.8: DiagnosticClearDtcLimitationEnum

Enumeration	**DiagnosticClearEventBehaviorEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticEvent
Note	Possible behavior for clearing events.
Literal	***Description***
noStatusByte Change	The event status byte keeps unchanged. **Tags:** atp.EnumerationValue=0
onlyThisCycleAnd Readiness	The OperationCycle and readiness bits of the event status byte are reset. **Tags:** atp.EnumerationValue=1

Table A.9: DiagnosticClearEventBehaviorEnum

Class	**DiagnosticComControl**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::CommunicationControl			
Note	This represents an instance of the "Communication Control" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticCommunicationControls			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
comControl Class	DiagnosticComControl Class	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticComControl in the given context.
customSub Function Number	PositiveInteger	0..1	attr	This attribute shall be used to define a custom sub-function number if none of the standardized values of category shall be used.

Table A.10: DiagnosticComControl

Class	«atpVariation» **DiagnosticCommonProps**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticCommonProps			
Note	This meta-class aggregates a number of common properties that are shared among a diagnostic extract. **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***

▽

△

Class	«atpVariation» **DiagnosticCommonProps**			
agingRequires TestedCycle	Boolean	1	attr	Defines whether the aging cycle counter is processed every aging cycles or else only tested aging cycle are considered. If the attribute is set to TRUE: only tested aging cycle are considered for aging cycle counter. If the attribute is set to FALSE: aging cycle counter is processed every aging cycle.
clearDtc Limitation	DiagnosticClearDtc LimitationEnum	1	attr	Defines the scope of the DEM_ClearDTC Api.
debounce AlgorithmProps	DiagnosticDebounce AlgorithmProps	*	aggr	Defines the used debounce algorithms relevant in the context of the enclosing DiagnosticCommonProps. Usually, there is a variety of debouncing algorithms to take into account and therefore the multiplicity of this aggregation is set to 0..*.
default Endianness	ByteOrderEnum	1	attr	Defines the default endianness of the data belonging to a DID or RID which is applicable if the DiagnosticData Element does not define the endianness via the swData DefProps.baseType attribute.
dtcStatus AvailabilityMask	PositiveInteger	1	attr	Mask for the supported DTC status bits by the Dem.
environment DataCapture	DiagnosticDataCapture Enum	0..1	attr	This attribute determines whether the capturing of environment data is done synchronously inside the report API function or whether the capturing shall be done asynchronously, i.e. after the report API function already terminated.
event Displacement Strategy	DiagnosticEvent DisplacementStrategy Enum	1	attr	This attribute defines, whether support for event displacement is enabled or not, and which displacement strategy is followed.
maxNumberOf EventEntries	PositiveInteger	0..1	attr	This attribute fixes the maximum number of event entries in the fault memory.
maxNumberOf Request Correctly Received Response Pending	PositiveInteger	1	attr	Maximum number of negative responses with response code 0x78 (requestCorrectlyReceived-ResponsePending) allowed per request. DCM will send a negative response with response code 0x10 (generalReject), in case the limit value gets reached. Value 0xFF means that no limit number of NRC 0x78 response apply.
memoryEntry StorageTrigger	DiagnosticMemoryEntry StorageTriggerEnum	1	attr	Describes the primary trigger to allocate an event memory entry.
occurrence Counter Processing	DiagnosticOccurrence CounterProcessing Enum	1	attr	This attribute defines the consideration of the fault confirmation process for the occurrence counter.
resetConfirmed BitOnOverflow	Boolean	1	attr	This attribute defines, whether the confirmed bit is reset or not while an event memory entry will be displaced.
responseOnAll RequestSids	Boolean	1	attr	If set to FALSE the DCM will not respond to diagnostic request that contains a service ID which is in the range from 0x40 to 0x7F or in the range from 0xC0 to 0xFF (Response IDs).

▽

△

Class	«atpVariation» **DiagnosticCommonProps**			
responseOn SecondDeclined Request	Boolean	1	attr	Defines the reaction upon a second request (ClientB) that can not be processed (e.g. due to priority assessment). TRUE: when the second request (Client B) can not be processed, it shall be answered with NRC21 BusyRepeat Request. FALSE: when the second request (Client B) can not be processed, it shall not be responded.
securityDelay TimeOnBoot	TimeValue	1	attr	Start delay timer on power on in seconds. This delay indicates the time at ECU boot power-on time where the Dcm remains in the default session and does not accept a security access.
statusBit HandlingTest FailedSinceLast Clear	DiagnosticStatusBit HandlingTestFailed SinceLastClearEnum	1	attr	This attribute defines, whether the aging and displacement mechanism shall be applied to the "Test FailedSinceLastClear" status bits.
statusBit StorageTest Failed	Boolean	1	attr	This parameter is used to activate/deactivate the permanent storage of the "TestFailed" status bits. true: storage activated false: storage deactivated
typeOfDtc Supported	DiagnosticTypeOfDtc SupportedEnum	1	attr	This attribute defines the format returned by Dem_Dcm GetTranslationType and does not relate to/influence the supported Dem functionality.
typeOfFreeze FrameRecord Numeration	DiagnosticTypeOf FreezeFrameRecord NumerationEnum	1	attr	This attribute defines the type of assigning freeze frame record numbers for event-specific freeze frame records.

Table A.11: DiagnosticCommonProps

Class	**DiagnosticConnectedIndicator**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticEvent			
Note	Description of indicators that are defined per DiagnosticEvent.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
behavior	DiagnosticConnected IndicatorBehaviorEnum	0..1	attr	Behavior of the linked indicator.
healingCycle	DiagnosticOperation Cycle	1	ref	The deactivation of indicators per event is defined as healing of a diagnostic event. The operation cycle in which the warning indicator will be switched off is defined here.
indicator	DiagnosticIndicator	1	ref	Reference to the used indicator.

Table A.12: DiagnosticConnectedIndicator

Class	**DiagnosticContributionSet**
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticContribution
Note	This meta-class represents a root node of a diagnostic extract. It bundles a given set of diagnostic model elements. The granularity of the DiagonsticContributionSet is arbitrary in order to support the aspect of decentralized configuration, i.e. different contributors can come up with an own DiagnosticContribution Set. **Tags:** atp.recommendedPackage=DiagnosticContributionSets

▽

△

Class	**DiagnosticContributionSet**			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
common Properties	DiagnosticCommon Props	0..1	aggr	This attribute represents a collection of diagnostic properties that are shared among the entire Diagnostic ContributionSet. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=commonProperties
element	DiagnosticCommon Element	*	ref	This represents a DiagnosticCommonElement considered in the context of the DiagnosticContributionSet **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element, variationPoint.shortLabel vh.latestBindingTime=postBuild
serviceTable	DiagnosticServiceTable	*	ref	This represents the collection of DiagnosticServiceTables to be considered in the scope of this Diagnostic ContributionSet. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceTable, variationPoint.short Label vh.latestBindingTime=postBuild

Table A.13: DiagnosticContributionSet

Class	**DiagnosticControlDTCSetting**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::ControlDTCSetting			
Note	This represents an instance of the "Control DTC Setting" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticControlDtcSettings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dtcSettingClass	DiagnosticControlDTC SettingClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the the reference represents the ability to access shared attributes among all DiagnosticControl DTCSetting in the given context.
dtcSetting Parameter	PositiveInteger	1	attr	This represents the DTCSettingType defined by ISO 14229-1. The pre-defined values are 1 (ON) and 2 (OFF).

Table A.14: DiagnosticControlDTCSetting

Class	***DiagnosticDataByIdentifier*** (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataByIdentifier			
Note	This represents an abstract base class for all diagnostic services that access data by identifier.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticReadDataByIdentifier, DiagnosticReadScalingDataByIdentifier, DiagnosticWriteDataBy Identifier			
Attribute	***Type***	***Mul.***	***Kind***	***Note***

▽

△

Class	***DiagnosticDataByIdentifier*** (abstract)			
dataIdentifier	DiagnosticAbstractData Identifier	1	ref	This represents the linked DiagnosticDataIdentifier.

Table A.15: DiagnosticDataByIdentifier

Class	**DiagnosticDataElement**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class represents the ability to describe a concrete piece of data to be taken into account for diagnostic purposes.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the meaning of the value of the array size.
maxNumberOf Elements	PositiveInteger	0..1	attr	The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take.
scalingInfoSize	PositiveInteger	0..1	attr	Size in bytes of scaling information for the DiagnosticData Element if used with DiagnosticReadScalingDataBy Identifier
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties in order to support the definition of e.g. computation formulae and data constraints.

Table A.16: DiagnosticDataElement

Class	**DiagnosticDataIdentifier**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class represents the ability to model a diagnostic data identifier (DID) that is fully specified regarding the payload at configuration-time. **Tags:** atp.recommendedPackage=DiagnosticDataIdentifiers			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticAbstractDataIdentifier*, *DiagnosticCommon Element*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataElement	DiagnosticParameter	1..*	aggr	This is the dataElement associated with the Diagnostic DataIdentifier. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataElement, variationPoint.short Label vh.latestBindingTime=postBuild
didSize	PositiveInteger	0..1	attr	This attribute indicates the size in bytes of the Diagnostic DataIdentifier.
representsVin	Boolean	0..1	attr	This attributes indicates whether the specific Diagnostic DataIdentifier represents the vehicle identification.
supportInfoByte	DiagnosticSupportInfo Byte	0..1	aggr	This attribute represents the supported information associated with the DiagnosticDataIdentifier.

Table A.17: DiagnosticDataIdentifier

Class	**DiagnosticDataIdentifierSet**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This represents the ability to define a list of DiagnosticDataIdentifiers that can be reused in different contexts. **Tags:** atp.recommendedPackage=DiagnosticDataIdentifierSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataIdentifier (ordered)	DiagnosticDataIdentifier	*	ref	Reference to an orderd list of Data Identifiers.

Table A.18: DiagnosticDataIdentifierSet

Class	**DiagnosticDebounceAlgorithmProps**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticDebouncingAlgorithm			
Note	Defines properties for the debounce algorithm class.			
Base	*ARObject*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
debounceAlgorithm	DiagEventDebounceAlgorithm	1	aggr	This represents the actual debounce algorithm.
debounceBehavior	DiagnosticDebounceBehaviorEnum	1	attr	This attribute defines how the event debounce algorithm will behave, if a related enable condition is not fulfilled or ControlDTCSetting of the related event is disabled.
debounceCounterStorage	Boolean	0..1	attr	Switch to store the debounce counter value non-volatile or not. true: debounce counter value shall be stored non-volatile false: debounce counter value is volatile

Table A.19: DiagnosticDebounceAlgorithmProps

Enumeration	**DiagnosticDebounceBehaviorEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticDebouncingAlgorithm
Note	Event debounce algorithm behavior options.
Literal	***Description***
freeze	The event debounce counter will be frozen with the current value and will not change while a related enable condition is not fulfilled or ControlDTCSetting of the related event is disabled. After all related enable conditions are fulfilled and ControlDTCSetting of the related event is enabled again, the event qualification will continue with the next report of the event (i.e. SetEventStatus). **Tags:** atp.EnumerationValue=0
reset	The event debounce counter will be reset to initial value if a related enable condition is not fulfilled or ControlDTCSetting of the related event is disabled. The qualification of the event will be restarted with the next valid event report. **Tags:** atp.EnumerationValue=1

Table A.20: DiagnosticDebounceBehaviorEnum

Class	**DiagnosticDemProvidedDataMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::ServiceMapping			
Note	This represents the ability to define the nature of a data access for a DiagnosticDataElement in the Dem. **Tags:** atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataElement	DiagnosticDataElement	0..1	ref	This represents the DiagnosticDataElement for which the access is further qualified by the DiagnosticDemProvided DataMapping.
dataProvider	NameToken	1	attr	This represents the ability to further specify the access within the Dem.

Table A.21: DiagnosticDemProvidedDataMapping

Class	**DiagnosticEcuReset**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EcuReset			
Note	This represents an instance of the "ECU Reset" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticEcuResets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
customSub Function Number	PositiveInteger	0..1	attr	This attribute shall be used to define a custom sub-function number if none of the standardized values of category shall be used.
ecuResetClass	DiagnosticEcuReset Class	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticEcuReset in the given context.

Table A.22: DiagnosticEcuReset

Class	**DiagnosticEcuResetClass**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EcuReset			
Note	This meta-class contains attributes shared by all instances of the "Ecu Reset" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticEcuResets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceClass*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
respondTo Reset	DiagnosticResponseTo EcuResetEnum	0..1	attr	This attribute defines whether the response to the Ecu Reset service shall be transmitted before or after the actual reset.

Table A.23: DiagnosticEcuResetClass

Class	**DiagnosticEnableCondition**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticCondition			
Note	Specification of an enable condition. **Tags:** atp.recommendedPackage=DiagnosticConditions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticCondition*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.24: DiagnosticEnableCondition

Class	***DiagnosticEnvCompareCondition*** (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EnvironmentalCondition			
Note	DiagnosticCompareConditions are atomic conditions. They are based on the idea of a comparison at runtime of some variable data with something constant. The type of the comparison (==, !=, <, <=, ...) is specified in DiagnosticCompareCondition.compareType.			
Base	*ARObject*, *DiagnosticEnvConditionFormulaPart*			
Subclasses	DiagnosticEnvDataCondition, DiagnosticEnvModeCondition			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
compareType	DiagnosticCompare TypeEnum	1	attr	This attributes represents the concrete type of the comparison.

Table A.25: DiagnosticEnvCompareCondition

Class	**DiagnosticEnvConditionFormula**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EnvironmentalCondition			
Note	A DiagnosticEnvConditionFormula embodies the computation instruction that is to be evaluated at runtime to determine if the DiagnosticEnvironmentalCondition is currently present (i.e. the formula is evaluated to true) or not (otherwise). The formula itself consists of parts which are combined by the logical operations specified by DiagnosticEnvConditionFormula.op. If a diagnostic functionality cannot be executed because an environmental condition fails then the diagnostic stack shall send a negative response code (NRC) back to the client. The value of the NRC is directly related to the specific formula and is therefore formalized in the attribute DiagnosticEnvCondition Formula.nrcValue.			
Base	*ARObject*, *DiagnosticEnvConditionFormulaPart*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
nrcValue	PositiveInteger	0..1	attr	This attribute represents the concrete NRC value that shall be returned if the condition fails.
op	DiagnosticLogical OperatorEnum	1	attr	This attribute represents the concrete operator (supported operators: and, or) of the condition formula.
part (ordered)	DiagnosticEnvCondition FormulaPart	*	aggr	This aggregation represents the collection of formula parts that can be combined by logical operators.

Table A.26: DiagnosticEnvConditionFormula

Class	DiagnosticEnvDataCondition			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EnvironmentalCondition			
Note	A DiagnosticEnvDataCondition is an atomic condition that compares the current value of the referenced DiagnosticDataElement with a constant value defined by the ValueSpecification. All compareTypes are supported.			
Base	*ARObject*, *DiagnosticEnvCompareCondition*, *DiagnosticEnvConditionFormulaPart*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
compareValue	ValueSpecification	1	aggr	This attribute represents a fixed compare value taken to evaluate the compare condition.
dataElement	DiagnosticDataElement	1	ref	This reference represents the related diagnostic data element.

Table A.27: DiagnosticEnvDataCondition

Class	DiagnosticEnvironmentalCondition			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EnvironmentalCondition			
Note	The meta-class DignosticEnvironmentalCondition formalizes the idea of a condition which is evaluated during runtime of the ECU by looking at "environmental" states (e.g. one such condition is that the vehicle is not driving, i.e. vehicle speed == 0). **Tags:** atp.recommendedPackage=DiagnosticEnvironmentalConditions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
formula	DiagnosticEnvCondition Formula	1	aggr	This attribute represents the formula part of the DiagnosticEnvironmentalCondition.
modeElement	DiagnosticEnvMode Element	*	aggr	This aggregation contains a representation of Mode Declarations in the context of a DiagnosticEnvironmental Condition.

Table A.28: DiagnosticEnvironmentalCondition

Class	DiagnosticEvent			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticEvent			
Note	This element is used to configure DiagnosticEvents. **Tags:** atp.recommendedPackage=DiagnosticEvents			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
clearEvent Behavior	DiagnosticClearEvent BehaviorEnum	0..1	attr	This attribute defines the resulting UDS DTC status byte for the related event, which shall not be cleared according to the ClearEventAllowed callback.
connected Indicator	DiagnosticConnected Indicator	*	aggr	Event specific description of Indicators. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
eventClear Allowed	DiagnosticEventClear AllowedEnum	0..1	attr	This attribute defines whether the Dem has access to a "ClearEventAllowed" callback.
eventFailure CycleCounter Threshold	PositiveInteger	0..1	attr	This attribute defines the number of failure cycles for the event based fault confirmation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild

▽

△

Class	**DiagnosticEvent**			
eventKind	DiagnosticEventKind Enum	1	attr	This attribute is used to distinguish between SWC and BSW events.
prestorage FreezeFrame	Boolean	1	attr	This attribute describes whether the Prestorage of Freeze Frames is supported by the assigned event or not. True: Prestorage of FreezeFrames is supported False: Prestorage of FreezeFrames is not supported
prestored Freezeframe StoredInNvm	Boolean	0..1	attr	If the Event uses a prestored freeze-frame (using the operations PrestoreFreezeFrame and ClearPrestored FreezeFrame of the service interface DiagnosticMonitor) this attribute indicates if the Event requires the data to be stored in non-volatile memory. TRUE = Dem shall store the prestored data in non-volatile memory, FALSE = Data can be lost at shutdown (not stored in Nvm)
recoverableIn SameOperation Cycle	Boolean	0..1	attr	If the attribute is set to true then reporting PASSED will reset the indication of a failed test in the current operation cycle. If the attribute is set to false then reporting PASSED will be ignored and not lead to a reset of the indication of a failed test.

Table A.29: DiagnosticEvent

Class	**DiagnosticEventToDebounceAlgorithmMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticMapping			
Note	Defines which Debounce Algorithm is applicable for a DiagnosticEvent. **Tags:** atp.recommendedPackage=DiagnosticMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
debounce Algorithm	DiagnosticDebounce AlgorithmProps	1	ref	Reference to a DebounceAlgorithm assigned to a DiagnosticEvent.
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which a Debounce Algorithm is assigned.

Table A.30: DiagnosticEventToDebounceAlgorithmMapping

Class	**DiagnosticEventToEnableConditionGroupMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticMapping			
Note	Defines which EnableConditionGroup is applicable for a DiagnosticEvent. **Tags:** atp.recommendedPackage=DiagnosticMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which an Enable ConditionGroup is assigned.
enableCondition Group	DiagnosticEnable ConditionGroup	1	ref	Reference to an EnableConditionGroup assigned to a DiagnosticEvent.

Table A.31: DiagnosticEventToEnableConditionGroupMapping

Class	**DiagnosticEventToOperationCycleMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticMapping			
Note	Defines which OperationCycle is applicable for a DiagnosticEvent. **Tags:** atp.recommendedPackage=DiagnosticMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which an Operation Cycle is assigned.
operationCycle	DiagnosticOperation Cycle	1	ref	Reference to an OperationCycle assigned to a Diagnostic Event.

Table A.32: DiagnosticEventToOperationCycleMapping

Class	**DiagnosticEventToStorageConditionGroupMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticMapping			
Note	Defines which StorageConditionGroup is applicable for a DiagnosticEvent. **Tags:** atp.recommendedPackage=DiagnosticMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which a Storage ConditionGroup is assigned.
storage ConditionGroup	DiagnosticStorage ConditionGroup	1	ref	Reference to a StorageConditionGroup assigned to a DiagnosticEvent.

Table A.33: DiagnosticEventToStorageConditionGroupMapping

Class	**DiagnosticEventToTroubleCodeUdsMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticMapping			
Note	Defines which UDS Diagnostic Trouble Code is applicable for a DiagnosticEvent. **Tags:** atp.recommendedPackage=DiagnosticMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which a UDS Diagnostic Trouble Code is assigned.
troubleCodeUds	DiagnosticTroubleCode Uds	1	ref	Reference to an UDS Diagnostic Trouble Code assigned to a DiagnosticEvent.

Table A.34: DiagnosticEventToTroubleCodeUdsMapping

Class	**DiagnosticExtendedDataRecord**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticExtendedDataRecord
Note	Description of an extended data record. **Tags:** atp.recommendedPackage=DiagnosticExtendedDataRecords
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*

▽

△

Class	**DiagnosticExtendedDataRecord**			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
customTrigger	String	0..1	attr	This attribute shall be taken to verbally describe the nature of the custom trigger.
recordElement	DiagnosticParameter	*	aggr	Defined DataElements in the extended record element.
recordNumber	PositiveInteger	1	attr	This attribute specifies an unique identifier for an extended data record.
trigger	DiagnosticRecord TriggerEnum	1	attr	This attribute specifies the primary trigger to allocate an event memory entry.
update	Boolean	1	attr	This attribute defines when an extended data record is captured. True: This extended data record is captured every time. False: This extended data record is only captured for new event memory entries.

Table A.35: DiagnosticExtendedDataRecord

Class	**DiagnosticFreezeFrame**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticFreezeFrame			
Note	This element describes combinations of DIDs for a non OBD relevant freeze frame. **Tags:** atp.recommendedPackage=DiagnosticFreezeFrames			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
customTrigger	String	0..1	attr	This attribute shall be taken to verbally describe the nature of the custom trigger.
recordNumber	PositiveInteger	0..1	attr	This attribute defines a record number for a freeze frame record. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
trigger	DiagnosticRecord TriggerEnum	1	attr	This attribute defines the primary trigger to allocate an event memory entry.
update	Boolean	0..1	attr	This attribute defines the approach when the freeze frame record is stored/updated. True: FreezeFrame record is captured every time. False: FreezeFrame record is only captured for new event memory entries.

Table A.36: DiagnosticFreezeFrame

Class	**DiagnosticIndicator**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticIndicator			
Note	Definition of an indicator. **Tags:** atp.recommendedPackage=DiagnosticIndicators			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
healingCycleCounterThreshold	PositiveInteger	1	attr	This attribute defines the number of healing cycles for the WarningIndicatorOffCriteria **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

▽

△

Class	DiagnosticIndicator			
type	DiagnosticIndicatorType Enum	0..1	attr	Defines the type of the indicator.

Table A.37: DiagnosticIndicator

Class	*DiagnosticMapping* (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticMapping			
Note	Abstract element for different kinds of diagnostic mappings.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticDemProvidedDataMapping, DiagnosticEventToDebounceAlgorithmMapping, DiagnosticEvent ToEnableConditionGroupMapping, DiagnosticEventToOperationCycleMapping, DiagnosticEventTo StorageConditionGroupMapping, DiagnosticEventToTroubleCodeJ1939Mapping, DiagnosticEventTo TroubleCodeUdsMapping, DiagnosticFimAliasEventGroupMapping, DiagnosticFimAliasEventMapping, DiagnosticInhibitSourceEventMapping, DiagnosticJ1939SpnMapping, DiagnosticServiceDataMapping, *DiagnosticSwMapping*, DiagnosticTroubleCodeUdsToClearConditionGroupMapping, DiagnosticTrouble CodeUdsToTroubleCodeObdMapping			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.38: DiagnosticMapping

Class	*DiagnosticMemoryDestination* (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This abstract meta-class represents a possible memory destination for a diagnostic event.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticMemoryDestinationMirror, DiagnosticMemoryDestinationPrimary, DiagnosticMemory DestinationUserDefined			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.39: DiagnosticMemoryDestination

Class	DiagnosticMemoryDestinationPrimary			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This represents a primary memory for a diagnostic event. **Tags:** atp.recommendedPackage=DiagnosticMemoryDestinations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMemory Destination*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.40: DiagnosticMemoryDestinationPrimary

Class	**DiagnosticMemoryDestinationUserDefined**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This represents a user-defined memory for a diagnostic event. **Tags:** atp.recommendedPackage=DiagnosticMemoryDestinations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMemoryDestination*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
memoryId	PositiveInteger	1	attr	This represents the identifier of the user-defined memory.

Table A.41: DiagnosticMemoryDestinationUserDefined

Class	**DiagnosticOperationCycle**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticOperationCycle			
Note	Definition of an operation cycle that is the base of the event qualifying and for Dem scheduling. **Tags:** atp.recommendedPackage=DiagnosticOperationCycles			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
automaticEnd	Boolean	1	attr	If set to true the driving cycle shall automatically end at either Dem_Shutdown() or Dem_Init().
cycleAutostart	Boolean	1	attr	This attribute defines if the operation cycles is automatically re-started during Dem_PreInit.
cycleStatus Storage	Boolean	1	attr	Defines if the operation cycle state is available over the power cycle (stored non-volatile) or not. • true: the operation cycle state is stored non-volatile • false: the operation cycle state is only stored volatile
type	DiagnosticOperation CycleTypeEnum	1	attr	Operation cycles types for the Dem.

Table A.42: DiagnosticOperationCycle

Class	**DiagnosticParameter**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class represents the ability to describe information relevant for the execution of a specific diagnostic service, i.e. it can be taken to parameterize the service.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
bitOffset	PositiveInteger	1	attr	This represents the bitOffset of the DiagnosticParameter
dataElement	DiagnosticDataElement	1	aggr	This represents the related dataElement of the Diagnostic Parameter **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
supportInfo	DiagnosticParameter SupportInfo	0..1	aggr	This attribute represents the ability to define which bit of the support info byte is representing this part of the PID.

Table A.43: DiagnosticParameter

Class	**DiagnosticReadDTCInformation**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::ReadDTCInformation			
Note	This represents an instance of the "Read DTC Information" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticReadDtcInformations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
read DTCInformation Class	DiagnosticReadDTC InformationClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticRead DTCInformation in the given context.

Table A.44: DiagnosticReadDTCInformation

Class	**DiagnosticReadDataByIdentifier**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataByIdentifier			
Note	This represents an instance of the "Read Data by Identifier" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticDataByIdentifiers			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticDataByIdentifier*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
readClass	DiagnosticReadDataBy IdentifierClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticReadDataBy Identifier in the given context.

Table A.45: DiagnosticReadDataByIdentifier

Class	**DiagnosticReadDataByIdentifierClass**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataByIdentifier			
Note	This meta-class contains attributes shared by all instances of the "Read Data by Identifier" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticDataByIdentifiers			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceClass*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
maxDidToRead	PositiveInteger	1	attr	This attribute represents the maximum number of allowed DIDs in a single instance of DiagnosticReadDataBy Identifier.

Table A.46: DiagnosticReadDataByIdentifierClass

Enumeration	**DiagnosticResponseToEcuResetEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EcuReset
Note	
Literal	***Description***
respondAfterReset	Answer to EcuReset service should come after the reset. **Tags:** atp.EnumerationValue=0
respondBefore Reset	Answer to EcuReset service should come before the reset. **Tags:** atp.EnumerationValue=1

Table A.47: DiagnosticResponseToEcuResetEnum

Class	**DiagnosticRoutine**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class represents the ability to define a diagnostic routine. **Tags:** atp.recommendedPackage=DiagnosticRoutines			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
id	PositiveInteger	1	attr	This is the numerical identifier used to identify the DiagnosticRoutine in the scope of diagnostic workflow **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
requestResult	DiagnosticRequest RoutineResults	0..1	aggr	This represents the ability to request the result of a running routine.
routineInfo	PositiveInteger	0..1	attr	This represents the routine info byte. The info byte contains a manufacturer-specific value (for the identification of record identifiers) that is reported to the tester. Other use cases for this attribute are mentioned in ISO 27145 and ISO 26021.
start	DiagnosticStartRoutine	0..1	aggr	This represents the ability to start a routine
stop	DiagnosticStopRoutine	0..1	aggr	This represents the ability to stop a running routine.

Table A.48: DiagnosticRoutine

Class	***DiagnosticRoutineSubfunction*** (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class acts as an abstract base class to routine subfunctions.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DiagnosticRequestRoutineResults, DiagnosticStartRoutine, DiagnosticStopRoutine			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
access Permission	DiagnosticAccess Permission	0..1	ref	This reference represents the access permission of the owning routine subfunction.

Table A.49: DiagnosticRoutineSubfunction

Class	**DiagnosticSecurityAccess**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::SecurityAccess			
Note	This represents an instance of the "Security Access" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticSecurityAccesss			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
requestSeedId	PositiveInteger	1	attr	This would be 0x01, 0x03, 0x05, ... The sendKey id can be computed by adding 1 to the requestSeedId
securityAccess Class	DiagnosticSecurity AccessClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticSecurityAccess in the given context.
securityLevel	DiagnosticSecurityLevel	1	ref	This reference identifies the applicable security level for the security access. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=securityLevel

Table A.50: DiagnosticSecurityAccess

Class	**DiagnosticSecurityAccessClass**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::SecurityAccess			
Note	This meta-class contains attributes shared by all instances of the "Security Access" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticSecurityAccesss			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceClass*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
sharedTimer	Boolean	0..1	attr	Switch between separate or single shared timer instance and timer value. * True: use shared timer instance and timer value for all security access levels combined. * False: use separate timer instance and timer values for each security level. **Tags:** atp.Status=draft

Table A.51: DiagnosticSecurityAccessClass

Class	**DiagnosticSecurityLevel**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm			
Note	This meta-class represents the ability to define a security level considered for diagnostic purposes. **Tags:** atp.recommendedPackage=DiagnosticSecurityLevels			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
accessData RecordSize	PositiveInteger	0..1	attr	This represents the size of the AccessDataRecord used in GetSeed. Unit:byte.

▽

△

Class	DiagnosticSecurityLevel			
keySize	PositiveInteger	1	attr	This represents the size of the security key. Unit: byte.
numFailed SecurityAccess	PositiveInteger	0..1	attr	This represents the number of failed security accesses after which the delay time is activated.
securityDelay Time	TimeValue	1	attr	This represents the delay time after a failed security access. Unit: second.
seedSize	PositiveInteger	1	attr	This represents the size of the security seed. Unit: byte.

Table A.52: DiagnosticSecurityLevel

Class	***DiagnosticServiceClass*** (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::CommonService			
Note	This meta-class provides the ability to define common properties that are shared among all instances of sub-classes of DiagnosticServiceInstance.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticClearDiagnosticInformationClass, DiagnosticClearResetEmissionRelatedInfoClass, DiagnosticComControlClass, DiagnosticControlDTCSettingClass, DiagnosticCustomServiceClass, DiagnosticDataTransferClass, DiagnosticDynamicallyDefineDataIdentifierClass, DiagnosticEcuResetClass, DiagnosticIoControlClass, DiagnosticReadDTCInformationClass, DiagnosticReadDataByIdentifierClass, DiagnosticReadDataByPeriodicIDClass, DiagnosticReadMemoryByAddressClass, DiagnosticReadScalingDataByIdentifierClass, DiagnosticRequestControlOfOnBoardDeviceClass, DiagnosticRequestCurrentPowertrainDataClass, DiagnosticRequestDownloadClass, DiagnosticRequestEmissionRelatedDTCClass, DiagnosticRequestEmissionRelatedDTCPermanentStatusClass, DiagnosticRequestFileTransferClass, DiagnosticRequestOnBoardMonitoringTestResultsClass, DiagnosticRequestPowertrainFreezeFrameDataClass, DiagnosticRequestUploadClass, DiagnosticRequestVehicleInfoClass, DiagnosticResponseOnEventClass, DiagnosticRoutineControlClass, DiagnosticSecurityAccessClass, DiagnosticSessionControlClass, DiagnosticTransferExitClass, DiagnosticWriteDataByIdentifierClass, DiagnosticWriteMemoryByAddressClass			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
access Permission	DiagnosticAccessPermission	0..1	ref	This represents the collection of DiagnosticAccessPermissions that allow for the execution of the referencing DiagnosticServiceClass.
access Permission Validity	DiagnosticAccessPermissionValidityEnum	1	attr	This attribute is responsible for clarifying the validity of the accessPermission reference.

Table A.53: DiagnosticServiceClass

Class	DiagnosticServiceDataIdentifierMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::DiagnosticDesign::DiagnosticMapping			
Note	This meta-class provides the ability to define a diagnostic access to an entire DID. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *DiagnosticSwMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticData Identifier	DiagnosticDataIdentifier	0..1	ref	This reference represents the applicable DiagnosticData Identfiier. **Tags:** atp.Status=draft
mappedSwc Service Dependency	SwcService Dependency	0..1	iref	**Tags:** atp.Status=draft

▽

△

Class	**DiagnosticServiceDataIdentifierMapping**			
process	ProcessDesign	1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft

Table A.54: DiagnosticServiceDataIdentifierMapping

Class	**DiagnosticServiceDataMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::ServiceMapping			
Note	This represents the ability to define a mapping of a diagnostic service to a software-component. This kind of service mapping is applicable for the usage of SenderReceiverInterfaces or event/notifier semantics in ServiceInterfaces on the adaptive platform. **Tags:** atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticData Element	DiagnosticDataElement	1	ref	This represents the applicable payload that corresponds to the referenced DataPrototype in the role mappedData Element or (in case of a usage on the adaptive platform) mappedApDataElement.
mappedApData Element	DataPrototype	0..1	iref	This represents the dataElement in the application software of an adaptive AUTOSAR application that is accessed for diagnostic purpose. **Tags:** atp.Status=draft
mappedData Element	DataPrototype	0..1	iref	This represents the dataElement in the application software that is accessed for diagnostic purpose. This role is applicable on the classic platform.
process	ProcessDesign	0..1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft

Table A.55: DiagnosticServiceDataMapping

Class	***DiagnosticServiceInstance*** (abstract)
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::CommonService
Note	This represents a concrete instance of a diagnostic service.
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*

▽

△

Class	***DiagnosticServiceInstance*** (abstract)			
Subclasses	DiagnosticClearDiagnosticInformation, DiagnosticClearResetEmissionRelatedInfo, DiagnosticCom Control, DiagnosticControlDTCSetting, DiagnosticCustomServiceInstance, *DiagnosticDataByIdentifier*, DiagnosticDynamicallyDefineDataIdentifier, DiagnosticEcuReset, DiagnosticIOControl, *Diagnostic MemoryByAddress*, DiagnosticReadDTCInformation, DiagnosticReadDataByPeriodicID, Diagnostic RequestControlOfOnBoardDevice, DiagnosticRequestCurrentPowertrainData, DiagnosticRequest EmissionRelatedDTC, DiagnosticRequestEmissionRelatedDTCPermanentStatus, DiagnosticRequestFile Transfer, DiagnosticRequestOnBoardMonitoringTestResults, DiagnosticRequestPowertrainFreezeFrame Data, DiagnosticRequestVehicleInfo, DiagnosticResponseOnEvent, DiagnosticRoutineControl, DiagnosticSecurityAccess, DiagnosticSessionControl			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
access Permission	DiagnosticAccess Permission	0..1	ref	This represents the collection of DiagnosticAccess Permissions that allow for the execution of the referencing DiagnosticServiceInstance..
serviceClass	DiagnosticServiceClass	0..1	ref	This represents the corresponding "class", i.e. this meta-class provides properties that are shared among all instances of applicable sub-classes of DiagnosticService Instance. The subclasses that affected by this pattern implement references to the applicable "class"-role that substantiate this abstract reference. **Stereotypes:** atpAbstract

Table A.56: DiagnosticServiceInstance

Class	**DiagnosticServiceSwMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::ServiceMapping			
Note	This represents the ability to define a mapping of a diagnostic service to a software-component or a basic-software module. If the former is used then this kind of service mapping is applicable for the usage of ClientServerInterfaces. **Tags:** atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *DiagnosticSwMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticData Element	DiagnosticDataElement	0..1	ref	This represents a DiagnosticDataElement required to execute the respective diagnostic service in the context of the diagnostic service mapping,
mappedBsw Service Dependency	BswService DependencyIdent	0..1	ref	This is supposed to represent a reference to a Bsw ServiceDependency. the latter is not derived from Referrable and therefore this detour needs to be implemented to still let BswServiceDependency become the target of a reference.
mappedFlatSwc Service Dependency	SwcService Dependency	0..1	ref	This represents the ability to refer to an AtomicSw ComponentType that is available without the definition of how it will be embedded into the component hierarchy.
mappedSwc Service DependencyIn Executable	SwcService Dependency	0..1	iref	This represents the ability to point into the component hiearchy of an adaptive AUTOSAR model (under possible consideration of the rootSoftwareComposition) **Tags:** atp.Status=draft
mappedSwc Service DependencyIn System	SwcService Dependency	0..1	iref	This represents the ability to point into the component hiearchy (under possible consideration of the root SoftwareComposition)

▽

△

Class	**DiagnosticServiceSwMapping**			
process	ProcessDesign	0..1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft
serviceInstance	DiagnosticService Instance	0..1	ref	This represents the service instance that needs to be considered in this diagnostics service mapping.

Table A.57: DiagnosticServiceSwMapping

Class	**DiagnosticSession**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm			
Note	This meta-class represents the ability to define a diagnostic session. **Tags:** atp.recommendedPackage=DiagnosticSessions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
id	PositiveInteger	1	attr	This is the numerical identifier used to identify the DiagnosticSession in the scope of diagnostic workflow
p2ServerMax	TimeValue	1	attr	This is the session value for P2ServerMax in seconds (per Session Control). The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds.
p2StarServer Max	TimeValue	1	attr	This is the session value for P2*ServerMax in seconds (per Session Control). The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds.

Table A.58: DiagnosticSession

Enumeration	**DiagnosticStatusBitHandlingTestFailedSinceLastClearEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticCommonProps
Note	Aging and displacement has no impact on the "TestFailedSinceLastClear" status bits.
Literal	***Description***
statusBitAgingAnd Displacement	**Tags:** atp.EnumerationValue=0
statusBitNormal	**Tags:** atp.EnumerationValue=1

Table A.59: DiagnosticStatusBitHandlingTestFailedSinceLastClearEnum

Class	**DiagnosticStorageCondition**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticCondition
Note	Specification of a storage condition. **Tags:** atp.recommendedPackage=DiagnosticConditions
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticCondition*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*

▽

△

Class	DiagnosticStorageCondition			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.60: DiagnosticStorageCondition

Class	DiagnosticTroubleCodeGroup			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	The diagnostic trouble code group defines the DTCs belonging together and thereby forming a group. **Tags:** atp.recommendedPackage=DiagnosticTroubleCodes			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dtc	DiagnosticTroubleCode	*	ref	This represents the collection of DiagnosticTroubleCodes defined by this DiagnosticTroubleCodeGroup. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dtc, variationPoint.shortLabel vh.latestBindingTime=postBuild
groupNumber	PositiveInteger	1	attr	This represents the base number of the DTC group. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table A.61: DiagnosticTroubleCodeGroup

Class	DiagnosticTroubleCodeProps			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This element defines common Dtc properties that can be reused by different non OBD-relevant DTCs. **Tags:** atp.recommendedPackage=DiagnosticTroubleCodePropss			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
aging	DiagnosticAging	0..1	ref	Reference to an aging algorithm in case that an aging/unlearning of the event is allowed.
agingAllowed	Boolean	0..1	attr	This represents the decision whether aging is allowed for this DiagnosticTroubleCodeProps.
environment CaptureTo Reporting	EnvironmentCaptureTo ReportingEnum	0..1	attr	This attribute determines the point in time, when the data actually is captured.
extendedData Record	DiagnosticExtended DataRecord	*	ref	Defines the links to an extended data class sampler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
freezeFrame	DiagnosticFreezeFrame	*	ref	Define the links to a freeze frame class sampler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
freezeFrame Content	DiagnosticDataIdentifier Set	0..1	ref	This represents the freeze frame layout as a set of DIDs.

▽

△

Class	**DiagnosticTroubleCodeProps**			
freezeFrame ContentWwh Obd	DiagnosticDataIdentifier Set	0..1	ref	This reefrence identifies the layout of the WWH-OBD freeze frame.
immediateNv DataStorage	Boolean	0..1	attr	Switch to enable immediate storage triggering of an according event memory entry persistently to NVRAM. true: immediate non-volatile storage triggering enabled false: immediate non-volatile storage triggering disabled
maxNumber FreezeFrame Records	PositiveInteger	0..1	attr	This attribute defines the number of according freeze frame records, which can maximal be stored for this event. Therefore all these freeze frame records have the same freeze frame class.
memory Destination	DiagnosticMemory Destination	*	ref	The event destination assigns events to none, one or multiple origins.
priority	PositiveInteger	1	attr	Priority of the event, in view of full event buffer. A lower value means higher priority.
significance	DiagnosticSignificance Enum	0..1	attr	Significance of the event, which indicates additional information concerning fault classification and resolution.

Table A.62: DiagnosticTroubleCodeProps

Class	**DiagnosticTroubleCodeUds**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This element is used to describe non OBD-relevant DTCs. **Tags:** atp.recommendedPackage=DiagnosticTroubleCodes			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticTroubleCode*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
considerPto Status	Boolean	0..1	attr	This attribute describes the affection of the event by the Dem PTO handling. True: the event is affected by the Dem PTO handling. False: the event is not affected by the Dem PTO handling.
dtcProps	DiagnosticTroubleCode Props	0..1	ref	Defined properties associated with the DemDTC.
eventObd Readiness Group	NameToken	0..1	attr	This attribute specifies the Event OBD Readiness group for PID $01 and PID $41 computation. This attribute is only applicable for emission-related ECUs.
functionalUnit	PositiveInteger	0..1	attr	This attribute specifies a 1-byte value which identifies the corresponding basic vehicle / system function which reports the DTC. This parameter is necessary for the report of severity information.
severity	DiagnosticUdsSeverity Enum	0..1	attr	DTC severity according to ISO 14229-1.
udsDtcValue	PositiveInteger	0..1	attr	Unique Diagnostic Trouble Code value for UDS.
wwhObdDtc Class	DiagnosticWwhObdDtc ClassEnum	0..1	attr	This attribute is used to identify (if applicable) the corresponding severity class of an WWH-OBD DTC.

Table A.63: DiagnosticTroubleCodeUds

Class	**DiagnosticTroubleCodeUdsToClearConditionGroupMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::DiagnosticDesign::DiagnosticClearCondition			
Note	This meta-class provides the ability to map a DiagnosticClearConditionGroup to a collection of Diagnostic TroubleCodeUds. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
clearCondition Group	DiagnosticClear ConditionGroup	0..1	ref	Thi reference identifies the applicable DiagnosticClear ConditionGroup. **Tags:** atp.Status=draft
troubleCodeUds	DiagnosticTroubleCode Uds	0..1	ref	This reference identifies the DiagnosticTroubleCodeUds that are relevant for the mapping. **Tags:** atp.Status=draft

Table A.64: DiagnosticTroubleCodeUdsToClearConditionGroupMapping

Class	**DiagnosticWriteDataByIdentifier**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataByIdentifier			
Note	This represents an instance of the "Write Data by Identifier" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticDataByIdentifiers			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticDataByIdentifier*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
writeClass	DiagnosticWriteDataBy IdentifierClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticWriteDataBy Identifier in the given context.

Table A.65: DiagnosticWriteDataByIdentifier

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractServiceInstance*, *AdaptiveModuleInstantiation*, AdaptiveSwcInternalBehavior, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *Atp Blueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanTp Address, CanTpChannel, CanTpNode, Chapter, CheckpointTransition, ClassContentConditional, ClientId Definition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *Comm ConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, *CryptoServiceMapping*, Data PrototypeGroup, DataTransformation, DdsRpcServiceDeployment, DependencyOnArtifact, Deterministic ClientResourceNeeds, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticData ▽

▽

△

Class	***Identifiable*** (abstract)			
	△ Element, DiagnosticFunctionInhibitSource, DiagnosticMasterToSlaveEventMapping, *DiagnosticRoutine Subfunction*, DoIpLogicAddress, E2EProfileConfiguration, ECUMapping, *EOCExecutableEntityRef Abstract*, EcuPartition, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, Ecuc EnumerationLiteralDef, EcucQuery, EcucValidationCondition, End2EndEventProtectionProps, EndToEnd Protection, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeature MapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeature Restriction, FMFeatureSelection, FieldMapping, FireAndForgetMapping, FlatInstanceDescriptor, Flexray ArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, General Parameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, Hw AttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToI PduMapping, ISignalTriggering, *IdentCaption*, InterfaceMapping, InternalTriggeringPoint, J1939Shared AddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, Mac MulticastGroup, McDataInstance, MemorySection, MethodMapping, ModeDeclaration, ModeDeclaration Mapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, NvBlockDescriptor, *Packageable Element*, ParameterAccess, PduToFrameMapping, PduTriggering, PerInstanceMemory, PersistencyFile Proxy, PersistencyKeyValuePair, PhmAction, *PhmActionItem*, PhmActionList, PhmArbitration, Phm LogicalExpression, PhmRule, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlot Mapping, ResourceConsumption, ResourceGroup, *RestAbstractEndpoint*, RestElementDef, Rest ResourceDef, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, Rpt Container, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptService Point, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, *SecureComProps*, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, *ServerCallPoint*, *ServiceEventDeployment*, *ServiceField Deployment*, ServiceInstanceToSignalMapping, *ServiceInterfaceElementMapping*, ServiceInterface ElementSecureComConfig, ServiceInterfaceMapping, *ServiceMethodDeployment*, *ServiceNeeds*, Signal BasedFieldToISignalTriggeringMapping, SocketAddress, SomeipEventGroup, SomeipProvidedEvent Group, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StartupConfig, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcTo ApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, TcpOptionFilter List, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtension Resource, TimingModeInstance, TlsCryptoCipherSuite, TlsJobMapping, Topic1, TpAddress, Traceable Text, *TracedFailure*, *TransformationProps*, TransformationPropsToServiceInterfaceElementMapping, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, Wait Point			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:** xml.sequenceOffset=-50
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25

▽

△

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

Table A.66: Identifiable

Class	**PPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
provided Interface	PortInterface	1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table A.67: PPortPrototype

Class	**RPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port requiring a certain port interface.			
Base	*ARObject*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
required Interface	PortInterface	1	tref	The interface that this port requires, i.e. the port depends on another port providing the specified interface. **Stereotypes:** isOfType

Table A.68: RPortPrototype

Class	**ServiceInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.Status=draft atp.recommendedPackage=ServiceInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime

Table A.69: ServiceInterface

Class	**SoftwareCluster**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage			
Note	This meta-class represents the ability to define an uploadable software-package, i.e. the SoftwareCluster shall contain all software and configuration for a given purpose. **Tags:** atp.Status=draft atp.recommendedPackage=SoftwareClusters			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
contained ARElement	ARElement	*	ref	This reference represents the collection of model elements that cannot derive from UploadablePackage Element and that contribute to the completeness of the definition of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName atp.Status=draft
containedFibex Element	FibexElement	*	ref	This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster. **Tags:** atp.Status=draft
contained Package Element	UploadablePackage Element	*	ref	This reference identifies model elements that are required to complete the manifest content. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=containedPackageElement atp.Status=draft
contained Process	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster. **Tags:** atp.Status=draft

▽

△

Class	**SoftwareCluster**			
dependsOn	SoftwareCluster Dependency	*	aggr	This aggregation can be taken to identify a dependency for the enclosing SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dependsOn atp.Status=draft
design	SoftwareClusterDesign	*	ref	This reference represents the identification of all Software ClusterDesigns applicable for the enclosing Software Cluster. **Stereotypes:** atpUriDef **Tags:** atp.Status=draft
diagnostic Address	SoftwareCluster DiagnosticAddress	*	aggr	This aggregation represents the collection of diagnostic addresses that apply for the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=diagnosticAddress atp.Status=draft
diagnostic Extract	DiagnosticContribution Set	0..1	ref	This reference represents the definition of the diagnostic extract applicable to the referencing SoftwareCluster **Tags:** atp.Status=draft
module Instantiation	AdaptiveModule Instantiation	*	ref	This reference identifies AdaptiveModuleInstantiations that need to be included with the SoftwareCluster in order to establish infrastructure required for the installation of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation atp.Status=draft
subSoftware Cluster	SoftwareCluster	*	ref	This reference is used to identify the sub-Software Clusters of an "umbrella" SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=subSoftwareCluster atp.Status=draft
version	String	1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster. The format of the version as well as how to tell a lower from a higher version is not prescribed by the AUTOSAR standard.

Table A.70: SoftwareCluster

Class	***SoftwareClusterDiagnosticAddress*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage			
Note	This meta-class represents the ability to define a diagnostic address in an abstract form. Sub-classes are supposed to clarify how the diagnostic address shall be defined according to the applicable addressing scheme (DoIP vs. CAN TP vs. ...). **Tags:** atp.Status=draft			
Base	*ARObject*			
Subclasses	SoftwareClusterDoipDiagnosticAddress			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
address Semantics	SoftwareCluster DiagnosticAddress SemanticsEnum	1	attr	This attribute clarifies whether the address value shall be interpreted as a physical or a functional address.

Table A.71: SoftwareClusterDiagnosticAddress

Enumeration	**SoftwareClusterDiagnosticAddressSemanticsEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage
Note	This meta-class defines a list of semantics for the interpretation of diagnostic addresses in the context of a SoftwareCluster. **Tags:** atp.Status=draft
Literal	*Description*
functionalAddress	This address represents a functional address. **Tags:** atp.EnumerationValue=1
physicalAddress	This address represents a physical address. **Tags:** atp.EnumerationValue=0

Table A.72: SoftwareClusterDiagnosticAddressSemanticsEnum

Class	«atpVariation» **SwDataDefProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the Data Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, sw AddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNative TypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

▽

△

Class	«atpVariation» **SwDataDefProps**			
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memory AllocationKeywordPolicy of the referenced SwAddr Method. **Tags:** xml.sequenceOffset=33
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70

▽

△

Class	«atpVariation» **SwDataDefProps**			
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290

▽

△

Class	«atpVariation» **SwDataDefProps**			
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
swValueBlock Size Mult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** xml.sequenceOffset=350
valueAxisData Type	ApplicationPrimitive DataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table A.73: SwDataDefProps

Class	**SwcServiceDependency**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	Specialization of ServiceDependency in the context of an SwcInternalBehavior. It allows to associate ports, port groups and (in special cases) data defined for an atomic software component to a given ServiceNeeds element.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceDependency*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
assignedData	RoleBasedData Assignment	*	aggr	Defines the role of an associated data object of the same component. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

▽

△

Class	SwcServiceDependency			
assignedPort	RoleBasedPort Assignment	*	aggr	Defines the role of an associated port of the same component. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedPort, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
representedPortGroup	PortGroup	0..1	ref	This reference specifies an association between the ServiceNeeeds and a PortGroup, for example to request a communication mode which applies for communication via these ports. The referred PortGroup shall be local to this atomic SWC, but via the links between the Port Groups, a tool can evaluate this information such that all the ports linked via this port group on the same ECU can be found.
serviceNeeds	ServiceNeeds	1	aggr	The associated ServiceNeeds.

Table A.74: SwcServiceDependency

B History of Constraints and Specification Items

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

B.1 Constraint and Specification Item History of this document according to AUTOSAR Release 17-10

B.1.1 Added Traceables in 17-10

Number	Heading
[SWS_DM_00277]	Cancellation of Active Protocol in case of External Service Processing
[SWS_DM_00278]	Cancellation of Active Protocol in case of Internal Processing
[SWS_DM_00279]	Cancellation of Active Protocol before Response Transmission
[SWS_DM_00280]	Cancellation of Active Protocol during Response Transmission
[SWS_DM_00281]	Cancellation of Active Protocol in Non-Default Session
[SWS_DM_00282]	Handling of `CurrentActiveProtocols`
[SWS_DM_00284]	SecurityAccess Service Interface
[SWS_DM_00286]	Configurable environmental condition check execution
[SWS_DM_00287]	Configurable environmental condition check criteria

▽

△

Number	Heading
[SWS_DM_00288]	Configurable environmental condition check evaluates to `TRUE`
[SWS_DM_00289]	Configurable environmental condition check evaluates to `FALSE`
[SWS_DM_00290]	Refusal of second diagnostic request from different diagnostic client without response
[SWS_DM_00291]	UdsMessage class
[SWS_DM_00292]	UdsMessage non public constructors
[SWS_DM_00293]	UdsMessage Address type
[SWS_DM_00294]	meta info map type
[SWS_DM_00295]	meta info map vendor type
[SWS_DM_00296]	TargetAddressType Address type
[SWS_DM_00297]	GetSa method
[SWS_DM_00298]	GetTa method
[SWS_DM_00299]	GetTaType method
[SWS_DM_00300]	GetPayload method readonly
[SWS_DM_00301]	GetPayload method
[SWS_DM_00302]	AddMetaInfo method
[SWS_DM_00303]	UdsMessage Pointer
[SWS_DM_00304]	Const UdsMessage Pointer
[SWS_DM_00305]	Const UdsMessage Pointer vendor type
[SWS_DM_00306]	UdsTransportProtocolMgr class
[SWS_DM_00307]	TransmissionResult type
[SWS_DM_00308]	Global Channel Identifier type
[SWS_DM_00309]	IndicateMessage method
[SWS_DM_00310]	NotifyMessageFailure method
[SWS_DM_00311]	HandleMessage method
[SWS_DM_00312]	TransmitConfirmation method
[SWS_DM_00313]	ChannelReestablished method
[SWS_DM_00314]	HandlerStopped method
[SWS_DM_00315]	UdsTransportProtocolHandler class
[SWS_DM_00316]	Header file
[SWS_DM_00317]	UdsTransportProtocolHandler constructor
[SWS_DM_00318]	UdsTransportProtocolHandler destructor
[SWS_DM_00319]	Initialize method
[SWS_DM_00320]	UdsTransportProtocolHandler UdsTransportProtocolMgr member
[SWS_DM_00321]	constructor member initialization
[SWS_DM_00322]	Start method
[SWS_DM_00323]	Stop method

▽

△

Number	Heading
[SWS_DM_00324]	UdsTransportProtocolHandler UdsTransportProtocolHandlerID member
[SWS_DM_00325]	GetHandlerID method
[SWS_DM_00326]	NotifyReestablishment method
[SWS_DM_00327]	Transmit method
[SWS_DM_00328]	UdsMessage Pointer vendor type
[SWS_DM_00329]	Lifecycle management of an `Uds Transport Protocol` implementation
[SWS_DM_00330]	Construction of an `Uds Transport Protocol` implementation
[SWS_DM_00331]	Initialization of an `Uds Transport Protocol` implementation
[SWS_DM_00332]	Starting of an `Uds Transport Protocol` implementation
[SWS_DM_00333]	Stopping of an `Uds Transport Protocol` implementation
[SWS_DM_00334]	UdsTransportProtocolMgr may be an abstract class
[SWS_DM_00335]	Header file
[SWS_DM_00336]	UdsTransportProtocolHandlerID
[SWS_DM_00337]	ChannelID
[SWS_DM_00338]	ByteVector
[SWS_DM_00339]	ByteVector vendor type
[SWS_DM_00340]	Waiting for Stop confirmation
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00342]	Indication of UDS message reception
[SWS_DM_00343]	Acceptance of UDS message reception
[SWS_DM_00344]	Refusal of UDS message reception
[SWS_DM_00345]	Forwarding of UDS message
[SWS_DM_00346]	Aborting of UDS message
[SWS_DM_00347]	Channel identification in Indication
[SWS_DM_00348]	Transmission of UDS response message
[SWS_DM_00349]	Reuse channel identifier of Indication
[SWS_DM_00350]	Confirmation of UDS message transmission
[SWS_DM_00351]	Confirmation Result
[SWS_DM_00356]	Requesting Notification of a channel reestablishment
[SWS_DM_00357]	Validity/lifetime of a Notification Request
[SWS_DM_00358]	Notification of a channel reestablishment
[SWS_DM_00359]	Persistent Storage of Notification Request
[SWS_DM_00360]	EcuReset positive response processing after reset
[SWS_DM_00361]	EcuReset application error processing
[SWS_DM_00362]	Checking Supported Subfunction for CompareKey
[SWS_DM_00363]	Positive response processing

▽

△

Number	Heading
[SWS_DM_00364]	Negative response processing
[SWS_DM_00365]	Suppression of response
[SWS_DM_00366]	Suppression of response for functional requests
[SWS_DM_00367]	No service processing
[SWS_DM_00368]	Sending busy responses
[SWS_DM_00369]	Max. number of busy responses
[SWS_DM_00370]	Support of UDS service ReadDTCInformation, Subfunction 0x06
[SWS_DM_00371]	Support of UDS service ReadDTCInformation, Subfunction 0x14
[SWS_DM_00372]	Support of UDS service ReadDTCInformation, Subfunction 0x17
[SWS_DM_00373]	Support of UDS service ReadDTCInformation, Subfunction 0x18
[SWS_DM_00374]	Support of UDS service ReadDTCInformation, Subfunction 0x19

Table B.1: Added Traceables in 17-10

B.1.2 Changed Traceables in 17-10

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00004]	Operation cycle persistency
[SWS_DM_00019]	Internal debounce counter incrementation
[SWS_DM_00020]	Internal debounce counter decrementation
[SWS_DM_00023]	Debounce counter jump down behavior
[SWS_DM_00030]	Calculation of the FDC based on the internal debounce timer
[SWS_DM_00042]	Canceling external service processors
[SWS_DM_00043]	Request refusal in case of no resources
[SWS_DM_00044]	Request refusal in case of non-default session active
[SWS_DM_00045]	Ignore ISO same resource access check
[SWS_DM_00046]	Each `Diagnostic Protocol` has own session resources
[SWS_DM_00047]	Each `Diagnostic Protocol` has own security-level resources
[SWS_DM_00048]	Request refusal in case of no resources
[SWS_DM_00049]	Refusal of second diagnostic request from different diagnostic client with BusyRepeatRequest
[SWS_DM_00051]	Cancellation of `Active Protocol` with lower priority
[SWS_DM_00052]	Selection between multiple cancellation candidates
[SWS_DM_00066]	Monitor initialization
[SWS_DM_00072]	Availability of enable condition service interfaces

▽

△

Number	Heading
[SWS_DM_00074]	Unsatisfied enable conditions
[SWS_DM_00088]	ControlDTCSetting influence
[SWS_DM_00089]	Reporting PREPASSED or PREFAILED for events without assigned debouncing algorithm
[SWS_DM_00096]	Validation Steps and Order
[SWS_DM_00098]	UDS message checks
[SWS_DM_00099]	Supported Service SID level checks
[SWS_DM_00100]	Supported Service subfunction level checks
[SWS_DM_00101]	Session Access SID level Permission
[SWS_DM_00102]	Session Access subfunction level Permission
[SWS_DM_00103]	Security Access level Permission
[SWS_DM_00105]	Configurable Manufacturer Permission Check Services
[SWS_DM_00106]	Signature of Manufacturer Permission Check Method
[SWS_DM_00107]	Configurable Supplier Permission Check Services
[SWS_DM_00108]	Signature of Supplier Permission Check Method
[SWS_DM_00111]	Configurable environment condition checks
[SWS_DM_00112]	Condition check definition
[SWS_DM_00136]	Request upload service processing
[SWS_DM_00148]	Persistent storage of event memory entries
[SWS_DM_00153]	Triggering for snapshot record storage
[SWS_DM_00156]	Triggering for extended data record storage and updates
[SWS_DM_00166]	Trigger to process event status
[SWS_DM_00167]	Ignoring reported events for not started operation cycles
[SWS_DM_00169]	Restart of operation cycles
[SWS_DM_00172]	Reaction on Unsupported DataIdentifier
[SWS_DM_00176]	External ReadDataByIdentifier processing
[SWS_DM_00177]	Negative Response processing
[SWS_DM_00179]	Positive Response processing
[SWS_DM_00180]	Provide Protocol Priority Configurability
[SWS_DM_00182]	Identification of a protocol for Priority Assignment
[SWS_DM_00184]	Protocol Match Search
[SWS_DM_00188]	Reaction on Unsupported DataIdentifier
[SWS_DM_00189]	WriteDataByIdentifier processing
[SWS_DM_00192]	Operation cycles are only ended once
[SWS_DM_00202]	Check for Supported RoutineIdentifier and Reaction
[SWS_DM_00203]	Check for Supported Subfunction and Reaction
[SWS_DM_00205]	Providing the VIN in DoIP protocol messages

▽

△

Number	Heading
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00249]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00252]	Reaction on Unsupported Subfunction
[SWS_DM_00258]	Cancellation of `Active Protocol` in non-default session
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00269]	Reaction on Unsupported Subfunction
[SWS_DM_00270]	Counting of attempts to change security level
[SWS_DM_00271]	Evaluate the number of failed security level change attempts
[SWS_DM_00272]	Expiration of the delay timer
[SWS_DM_00273]	Notification event upon `snapshot record` updates
[SWS_DM_00274]	Definition of an active diagnostic protocol

Table B.2: Changed Traceables in 17-10

B.1.3 Deleted Traceables in 17-10

Number	Heading
[SWS_DM_00001]	Availability of operation cycle service interfaces
[SWS_DM_00053]	Cancellation of Active Protocol
[SWS_DM_00054]	Generic UDS Service Interface
[SWS_DM_00073]	Checking enable conditions after status reports
[SWS_DM_00075]	Fulfilled enable conditions
[SWS_DM_00076]	Checking storage conditions in case the storage of event-related data is triggered
[SWS_DM_00077]	Checking storage conditions in case the update of event-related data is triggered
[SWS_DM_00081]	Routine Service Interface
[SWS_DM_00093]	Service Validation Interface
[SWS_DM_00094]	Data Services Interface
[SWS_DM_00149]	DTC related data
[SWS_DM_00157]	Snapshot record record data layout
[SWS_DM_00171]	Check for Supported DataIdentifier
[SWS_DM_00187]	Check for Supported DataIdentifier
[SWS_DM_00204]	Reaction on Unsupported Subfunction
[SWS_DM_00251]	Check for Supported Subfunction
[SWS_DM_CONSTR_00275]	Response processing after the actual reset

Table B.3: Deleted Traceables in 17-10

B.1.4 Added Constraints in 17-10

none

B.1.5 Changed Constraints in 17-10

none

B.1.6 Deleted Constraints in 17-10

none

B.2 Constraint and Specification Item History of this document according to AUTOSAR Release 18-03

B.2.1 Added Traceables in 18-03

Number	Heading
[SWS_DM_00001]	SRS Diagnostics
[SWS_DM_00376]	Positive response processing
[SWS_DM_00377]	Enable condition influence on debouncing behavior (reset)
[SWS_DM_00378]	ControlDTCSetting influence (reset)
[SWS_DM_00379]	Handling of storage conditions
[SWS_DM_00380]	Support for S3 timer
[SWS_DM_00381]	Session timeout
[SWS_DM_00382]	Session timeout start
[SWS_DM_00383]	Session timeout stop
[SWS_DM_00384]	IndicationResult type
[SWS_DM_00385]	Acceptance of UDS message reception
[SWS_DM_00386]	Ignoring UDS message reception because DM is busy
[SWS_DM_00387]	Ignoring UDS message reception because DM has no (memory) ressources
[SWS_DM_00388]	Filling provided UdsMessage
[SWS_DM_00389]	Skipping Forwarding of UDS message
[SWS_DM_00390]	Dispatching physical Request
[SWS_DM_00391]	Dispatching functional Request
[SWS_DM_00392]	Properties of returned UdsMessage

▽

△

Number	Heading
[SWS_DM_00393]	Retrieving data for `internal` `DiagnosticDataElement`s
[SWS_DM_00397]	Retrieving data for `external` `DiagnosticDataElement`s
[SWS_DM_00401]	Reading Diagnostic Data Identifier on Data Element level
[SWS_DM_00402]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00403]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00404]	Default Service Interface for reading `DiagnosticDataIdentifier`
[SWS_DM_00405]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00406]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00407]	Default Service Interface for writing `DiagnosticDataIdentifier`
[SWS_DM_00408]	Retrieving data for requested DataIdentifier
[SWS_DM_00409]	Check supported DataIdentifier
[SWS_DM_00410]	Check session permission
[SWS_DM_00411]	Check security level permission
[SWS_DM_00412]	Check requested number of DataIdentifiers
[SWS_DM_00413]	Check supported DataIdentifier in active session
[SWS_DM_00414]	Check supported DataIdentifier on active security level
[SWS_DM_00415]	Check supported DataIdentifier
[SWS_DM_00416]	Check supported DataIdentifier in active session
[SWS_DM_00417]	Check supported DataIdentifier on active security level
[SWS_DM_00418]	Writing data for requested DataIdentifier
[SWS_DM_00419]	Reaction on ApplicationError
[SWS_DM_00420]	Instantiation of Diagnostic Server
[SWS_DM_00434]	Providing the `PowerMode` in DoIP protocol messages
[SWS_DM_CONSTR_00394]	`internal` `DiagnosticDataElement`s are read-only
[SWS_DM_CONSTR_00395]	Restriction on `DEM-exclusive` `DiagnosticDataElement`s
[SWS_DM_CONSTR_00396]	Restriction on `DCM-exclusive` `DiagnosticDataElement`s

Table B.4: Added Traceables in 18-03

B.2.2 Changed Traceables in 18-03

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00005]	DoIP Support
[SWS_DM_00007]	Uniqueness of diagnostic events
[SWS_DM_00008]	Diagnostic event processing interface

▽

△

Number	Heading
[SWS_DM_00012]	DoIP configurable source address identification
[SWS_DM_00013]	Events without debouncing
[SWS_DM_00014]	Use of counter-based debouncing for events
[SWS_DM_00015]	Use of timer based debouncing for events
[SWS_DM_00017]	Calculation of the FDC based on the internal debounce counter
[SWS_DM_00018]	Internal debounce counter init and storage
[SWS_DM_00019]	Internal debounce counter incrementation
[SWS_DM_00020]	Internal debounce counter decrementation
[SWS_DM_00021]	Direct failed qualification of counter-based events
[SWS_DM_00022]	Debounce counter jump up behavior
[SWS_DM_00023]	Debounce counter jump down behavior
[SWS_DM_00024]	Qualified failed event using counter-based debouncing
[SWS_DM_00025]	Qualified passed event using counter-based debouncing
[SWS_DM_00026]	Application resetting the debounce counter
[SWS_DM_00028]	Debounce counter persistency
[SWS_DM_00029]	Direct passed qualification of counter-based events
[SWS_DM_00030]	Calculation of the FDC based on the internal debounce timer
[SWS_DM_00031]	Starting time-based event debouncing for failed
[SWS_DM_00032]	Restrictions on restarting a running event debounce timer for failed
[SWS_DM_00033]	Debounce timer behavior upon reported failed
[SWS_DM_00034]	Starting time-based event debouncing for passed
[SWS_DM_00035]	Restrictions on restarting a running event debounce timer for passed
[SWS_DM_00036]	Debounce timer behavior upon reported passed
[SWS_DM_00037]	Debounce time freeze request
[SWS_DM_00038]	Continuing a frozen debounce timer
[SWS_DM_00039]	Resetting the debounce counter upon starting or restarting an operation cycle
[SWS_DM_00040]	Definition of debounce counter reset
[SWS_DM_00041]	Behavior according to ISO Multiple client handling flow
[SWS_DM_00042]	Cancelling external service processors
[SWS_DM_00043]	Request refusal in case of no resources
[SWS_DM_00044]	Request refusal in case of non-default session active
[SWS_DM_00045]	Ignore ISO same resource access check
[SWS_DM_00046]	Each `Diagnostic Protocol` has own session resources
[SWS_DM_00047]	Each `Diagnostic Protocol` has own security-level resources
[SWS_DM_00048]	Request refusal in case of no resources
[SWS_DM_00049]	Refusal of second diagnostic request from different diagnostic client with BusyRepeatRequest

▽

△

Number	Heading
[SWS_DM_00052]	Selection between multiple cancellation candidates
[SWS_DM_00055]	Supported event memories
[SWS_DM_00057]	Availability of a user-defined event memory
[SWS_DM_00058]	DTC interpretation format
[SWS_DM_00060]	Set of supported DTCs
[SWS_DM_00061]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCByStatusMask
[SWS_DM_00062]	Mapping between ISO 14229-1[1] and Autosar Diagnostic Extract Template [2] of the DTCFormatIdentifier
[SWS_DM_00063]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord
[SWS_DM_00064]	Definition of DTC groups
[SWS_DM_00065]	Always supported availability of the group of all DTCs
[SWS_DM_00069]	Monitor initialization for enable condition reenabling reason
[SWS_DM_00070]	Monitor initialization for DTC setting re-enabling reason
[SWS_DM_00071]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00074]	Handling of enable conditions
[SWS_DM_00085]	Internal debounce counter init
[SWS_DM_00086]	Resetting the debounce counter after clearing DTC
[SWS_DM_00087]	Enable condition influence on debouncing behavior (freeze)
[SWS_DM_00088]	ControlDTCSetting influence (freeze)
[SWS_DM_00089]	Reporting PREPASSED or PREFAILED for events without assigned debouncing algorithm
[SWS_DM_00090]	Support of UDS service ClearDiagnosticInformation
[SWS_DM_00091]	Evaluation of ClearDiagnosticInformation parameters
[SWS_DM_00092]	Parameter range check for groupOfDTC request parameter
[SWS_DM_00096]	Validation Steps and Order
[SWS_DM_00097]	Abort on failed verification step
[SWS_DM_00111]	Configurable environment condition checks
[SWS_DM_00112]	Condition check definition
[SWS_DM_00113]	Positive response for UDS service 0x14
[SWS_DM_00114]	Limitation to one simultaneous DTC clear operation
[SWS_DM_00115]	Memory error handling while clearing DTCs
[SWS_DM_00116]	Clearing a DTC group
[SWS_DM_00117]	Clearing a DTC
[SWS_DM_00118]	Event specific configuration to allow clearing of a DTC
[SWS_DM_00119]	Init value for events with clear allowed information

▽

△

Number	Heading
[SWS_DM_00120]	Description of application interface to control the clear event behavior
[SWS_DM_00121]	Forbidden clearing of snapshot records and extended data records
[SWS_DM_00122]	UDS response behavior on not allowed clear operations
[SWS_DM_00123]	Block status byte clearing during a clear DTC operation
[SWS_DM_00124]	Limited status byte clearing during a clear DTC operation
[SWS_DM_00125]	Linking between event clear allowed and clearing a DTC
[SWS_DM_00128]	Realisation of UDS service 0x34 RequestDownload
[SWS_DM_00129]	Supported addressAndLengthFormatIdentifier
[SWS_DM_00130]	Not supported addressAndLengthFormatIdentifier
[SWS_DM_00136]	Request upload service processing
[SWS_DM_00138]	Transfer data service processing
[SWS_DM_00139]	Transfer data service validation
[SWS_DM_00142]	Transfer data service processing
[SWS_DM_00143]	Transfer data service validation
[SWS_DM_00144]	Parallel clearing DTCs in different `DiagnosticMemoryDestination`
[SWS_DM_00145]	Allow only one simultaneous clear DTC operation for one `DiagnosticMemoryDestination`
[SWS_DM_00146]	Unlock clear DTC operation for one `DiagnosticMemoryDestination`
[SWS_DM_00147]	Behavior while trying to clear DTCs on a locked `DiagnosticMemoryDestination`
[SWS_DM_00148]	Persistent storage of event memory entries
[SWS_DM_00151]	Snapshot record numeration
[SWS_DM_00152]	Number of snapshot records for a DTC
[SWS_DM_00153]	Triggering for snapshot record storage
[SWS_DM_00154]	Number of extended data for a DTC
[SWS_DM_00155]	Extended data record numeration
[SWS_DM_00156]	Triggering for extended data record storage and updates
[SWS_DM_00159]	Allow only to clear `GroupOfAllDTCs`
[SWS_DM_00160]	Allow to clear single `DTCs`
[SWS_DM_00161]	Negative response on not supported GroupOfDTC parameter
[SWS_DM_00162]	Point in time for positive response for ClearDTC
[SWS_DM_00166]	Trigger to process event status
[SWS_DM_00167]	Ignoring reported events for not started operation cycles
[SWS_DM_00168]	Availability of `DiagnosticMonitor` service interfaces
[SWS_DM_00177]	Reaction on ApplicationError
[SWS_DM_00180]	Provide Protocol Priority Configurability
[SWS_DM_00182]	Identification of a protocol for Priority Assignment

▽

△

Number	Heading
[SWS_DM_00183]	Wildcards per attribute
[SWS_DM_00184]	Protocol Match Search
[SWS_DM_00194]	Definition of the user-defined fault memory number for ClearDiagnosticInformation
[SWS_DM_00202]	Check for Supported RoutineIdentifier and Reaction
[SWS_DM_00203]	Check for Supported Subfunction and Reaction
[SWS_DM_00205]	Providing the VIN in DoIP protocol messages
[SWS_DM_00213]	DTC status processing
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00217]	DTC status bit transitions triggered by ClearDiagnosticInformation UDS service
[SWS_DM_00218]	Confirmation
[SWS_DM_00219]	Observability of the status byte
[SWS_DM_00220]	Notification about the changes of the status byte
[SWS_DM_00223]	Handling of 'warningIndicatorRequested' bit
[SWS_DM_00227]	Check for supported sessions
[SWS_DM_00229]	Support of UDS service ControlDTCSetting
[SWS_DM_00230]	Check for supported subfunctions
[SWS_DM_00231]	Invalid value for optional request parameter
[SWS_DM_00232]	Support of Subfunction 0x01 (ON)
[SWS_DM_00233]	Support of Subfunction 0x02 (OFF)
[SWS_DM_00236]	Realization of UDS service 0x27 SecurityAccess
[SWS_DM_00237]	Aging
[SWS_DM_00238]	Aging and healing
[SWS_DM_00239]	Aging counter
[SWS_DM_00240]	Processing the aging counter
[SWS_DM_00241]	Aging cycle and threshold
[SWS_DM_00242]	Reoccurrence after aging
[SWS_DM_00243]	Aging-related UDS status byte processing
[SWS_DM_00244]	Support of UDS service ReadDTCInformation, Subfunction 0x01
[SWS_DM_00245]	Support of UDS service ReadDTCInformation, Subfunction 0x02
[SWS_DM_00246]	Support of UDS service ReadDTCInformation, Subfunction 0x04
[SWS_DM_00247]	Support of UDS service ReadDTCInformation, Subfunction 0x07
[SWS_DM_00248]	Notification about session change
[SWS_DM_00249]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00250]	Notification about security-level change

▽

△

Number	Heading
[SWS_DM_00258]	Cancellation of `Active Protocol` in non-default session
[SWS_DM_00259]	Completion of already `Active Protocol`s in default session
[SWS_DM_00260]	instances of interface ClearDTC
[SWS_DM_00261]	Usage of ClearDTC Interface
[SWS_DM_00262]	Common semantic behavior for ClearDTC triggered via diagnostics or application
[SWS_DM_00265]	ClearDTC called while another clear operation is in progress
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00270]	Counting of attempts to change security level
[SWS_DM_00271]	Evaluate the number of failed security level change attempts
[SWS_DM_00272]	Expiration of the delay timer
[SWS_DM_00273]	Notification event upon `snapshot record` updates
[SWS_DM_00277]	Cancellation of Active Protocol in case of External Service Processing
[SWS_DM_00278]	Cancellation of Active Protocol in case of Internal Processing
[SWS_DM_00279]	Cancellation of Active Protocol before Response Transmission
[SWS_DM_00280]	Cancellation of Active Protocol at Response Transmission
[SWS_DM_00281]	Cancellation of active DiagnosticConversation in Non-Default Session
[SWS_DM_00282]	Handling of non-/active diagnostic conversations
[SWS_DM_00286]	Configurable environmental condition check execution
[SWS_DM_00290]	Refusal of second diagnostic request from different diagnostic client without response
[SWS_DM_00309]	IndicateMessage method
[SWS_DM_00316]	Header file
[SWS_DM_00329]	Lifecycle management of an `Uds Transport Protocol` implementation
[SWS_DM_00330]	Construction of an `Uds Transport Protocol` implementation
[SWS_DM_00331]	Initialization of an `Uds Transport Protocol` implementation
[SWS_DM_00332]	Starting of an `Uds Transport Protocol` implementation
[SWS_DM_00333]	Stopping of an `Uds Transport Protocol` implementation
[SWS_DM_00335]	Header file
[SWS_DM_00340]	Waiting for Stop confirmation
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00342]	Indication of UDS message reception
[SWS_DM_00345]	Forwarding of UDS message
[SWS_DM_00346]	Aborting of UDS message
[SWS_DM_00347]	Channel identification in Indication
[SWS_DM_00348]	Transmission of UDS response message
[SWS_DM_00349]	Reuse channel identifier of Indication

▽

△

Number	Heading
[SWS_DM_00350]	Confirmation of UDS message transmission
[SWS_DM_00351]	Confirmation Result
[SWS_DM_00356]	Requesting Notification of a channel reestablishment
[SWS_DM_00357]	Validity/lifetime of a Notification Request
[SWS_DM_00358]	Notification of a channel reestablishment
[SWS_DM_00359]	Persistent Storage of Notification Request
[SWS_DM_00362]	Checking Supported Subfunction for CompareKey
[SWS_DM_00363]	Unsupported Subfunction
[SWS_DM_00366]	Suppression of response for functional requests
[SWS_DM_00369]	Max. number of busy responses
[SWS_DM_00370]	Support of UDS service ReadDTCInformation, Subfunction 0x06
[SWS_DM_00371]	Support of UDS service ReadDTCInformation, Subfunction 0x14
[SWS_DM_00372]	Support of UDS service ReadDTCInformation, Subfunction 0x17
[SWS_DM_00373]	Support of UDS service ReadDTCInformation, Subfunction 0x18
[SWS_DM_00374]	Support of UDS service ReadDTCInformation, Subfunction 0x19
[SWS_DM_CONSTR_00083]	Restriction on supported DTC format
[SWS_DM_CONSTR_00088]	Restriction on the configuration of the DTC group GroupOfAllDTCs
[SWS_DM_CONSTR_00094]	Each DTC shall be assigned to an event memory destination
[SWS_DM_CONSTR_00188]	Required operation cycles for diagnostic events
[SWS_DM_CONSTR_00206]	Supported format for data identifier for VINDataIdentifier
[SWS_DM_CONSTR_00207]	Required VINDataIdentifier

Table B.5: Changed Traceables in 18-03

B.2.3 Deleted Traceables in 18-03

Number	Heading
[SWS_DM_00072]	Availability of enable condition service interfaces
[SWS_DM_00078]	Unsatisfied storage conditions
[SWS_DM_00079]	Fulfilled storage conditions
[SWS_DM_00172]	Reaction on Unsupported DataIdentifier
[SWS_DM_00173]	Classification as Internally implemented DID
[SWS_DM_00174]	Internally implemented DID ActiveDiagnosticSessionDataIdentifier
[SWS_DM_00175]	Classification as Externally implemented DID
[SWS_DM_00176]	External ReadDataByIdentifier processing
[SWS_DM_00178]	Check requested number of DataIdentifiers
[SWS_DM_00179]	Positive Response processing

▽

△

Number	Heading
[SWS_DM_00188]	Reaction on Unsupported DataIdentifier
[SWS_DM_00189]	WriteDataByIdentifier processing
[SWS_DM_00190]	Negative Response processing
[SWS_DM_00191]	Positive Response processing
[SWS_DM_00264]	ClearDTC call on invalid DTCOrigin
[SWS_DM_00292]	UdsMessage non public constructors
[SWS_DM_00343]	Acceptance of UDS message reception
[SWS_DM_00344]	Refusal of UDS message reception

Table B.6: Deleted Traceables in 18-03

B.2.4 Added Constraints in 18-03

none

B.2.5 Changed Constraints in 18-03

none

B.2.6 Deleted Constraints in 18-03

none

B.3 Constraint and Specification Item History of this document according to AUTOSAR Release 18-10

B.3.1 Added Traceables in 18-10

Number	Heading
[SWS_DM_00421]	Identification of a Diagnostic Client
[SWS_DM_00422]	Instantiation of Diagnostic Conversation Service Interface
[SWS_DM_00423]	Assignment of Diagnostic Conversation to Service Instances
[SWS_DM_00424]	Reset Service Instance fields on end of Diagnostic Conversation
[SWS_DM_00425]	Procedure to assign UDS requests to Diagnostic Conversations
[SWS_DM_00426]	Assigning a UDS request to an existing Diagnostic Conversation

▽

△

Number	Heading
[SWS_DM_00427]	Priority of a Diagnostic Conversation
[SWS_DM_00428]	Treatment of priority values
[SWS_DM_00429]	Prioritization in case of Pseudo Parallel Mode and active non-default session
[SWS_DM_00430]	Prioritization against all Diagnostic Conversations
[SWS_DM_00431]	Replacement of Diagnostic Conversations
[SWS_DM_00432]	Initial values for Diagnostic Conversation
[SWS_DM_00433]	Refusal of diagnostic request due to busy Diagnostic Conversation
[SWS_DM_00435]	Default session change trigger from AAs
[SWS_DM_00436]	Providing the GID in DoIP protocol messages
[SWS_DM_00437]	Check supported RoutineIdentifier on active security level
[SWS_DM_00438]	Check Support of UDS service RequestUpload (0x35) in active session
[SWS_DM_00439]	Check Support of UDS service RequestUpload (0x35) on active security level
[SWS_DM_00440]	Check Support of UDS service TransferData (0x36) in active session
[SWS_DM_00441]	Check Support of UDS service TransferData (0x36) on active security level
[SWS_DM_00442]	Check Support of UDS service RequestTransferExit (0x37) in active session
[SWS_DM_00443]	Check Support of UDS service RequestTransferExit (0x37) on active security level
[SWS_DM_00444]	Check Support of UDS service ControlDTCSetting (0x85) in active session
[SWS_DM_00445]	Check Support of UDS service ControlDTCSetting (0x85) on active security level
[SWS_DM_00446]	Check Support of UDS service RequestDownload (0x34) in active session
[SWS_DM_00447]	Check Support of UDS service RequestDownload (0x34) on active security level
[SWS_DM_00448]	Check supported RoutineIdentifier in active session
[SWS_DM_00449]	Supported DoIP message types
[SWS_DM_00451]	
[SWS_DM_00452]	
[SWS_DM_00475]	DoIP Version
[SWS_DM_00476]	User Controlled Warning IndicatorRequest-bit
[SWS_DM_00477]	Not Storing of 'warningIndicatorRequested' bit
[SWS_DM_00478]	Persistent Storage of failed attempts to change security level
[SWS_DM_00479]	Blocking Timer for security access on Restart or Power down - power up cycle
[SWS_DM_00480]	Security Access Blocking Timer
[SWS_DM_00481]	Handling of DiagnosticClearConditions
[SWS_DM_00482]	Cancellation of a Diagnostic Conversation
[SWS_DM_00483]	Cancellation trigger from AAs
[SWS_DM_00484]	Updating DiagnosticConversation Service Instance fields

▽

△

Number	Heading
[SWS_DM_00485]	Reinitialization of Service Instance on Cancellation of a Diagnostic Conversation
[SWS_DM_00487]	Ignoring UDS message reception because of unknown target address
[SWS_DM_00491]	Realisation of UDS service 0x86 ResponseOnEvent
[SWS_DM_00492]	Client Server communication
[SWS_DM_00493]	Reestablishing of Client Server communication
[SWS_DM_00494]	Supported sub functions of ResponseOnEvent service
[SWS_DM_00495]	Start initialisation of ResponseOnEvent
[SWS_DM_00496]	Stop initialisation of ResponseOnEvent
[SWS_DM_00497]	Clear initialisation of ResponseOnEvent
[SWS_DM_00498]	Exclusive ResponseOnEvent ressources
[SWS_DM_00499]	Replacement of a not started ResponseOnEvent initialisation
[SWS_DM_00500]	Replacement of a started ResponseOnEvent initialisation
[SWS_DM_00501]	Behavior while trying ResponseOnEvent activation while ResponseOnEvent is not initialised
[SWS_DM_00503]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00504]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00505]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00506]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00507]	Length check on UDS Service 0x27 request with Subfunction for RequestSeed
[SWS_DM_00508]	Reading DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00509]	Writing DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00651]	NumberOfStoredEntries
[SWS_DM_09010]	
[SWS_DM_09012]	
[SWS_DM_09015]	
[SWS_DM_09016]	
[SWS_DM_09017]	
[SWS_DM_09021]	
[SWS_DM_09028]	
[SWS_DM_CONSTR_00288]	Delay time value for sharedTimer
[SWS_DM_NA]	

Table B.7: Added Traceables in 18-10

B.3.2 Changed Traceables in 18-10

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00004]	Operation cycle persistency
[SWS_DM_00005]	DoIP Support
[SWS_DM_00008]	Diagnostic event processing interface
[SWS_DM_00011]	Selectability of parallelism mode
[SWS_DM_00014]	Use of counter-based debouncing for events
[SWS_DM_00015]	Use of timer based debouncing for events
[SWS_DM_00016]	Configurable number of supported parallel Diagnostic Conversations
[SWS_DM_00020]	Internal debounce counter decrementation
[SWS_DM_00026]	Application resetting the debounce counter
[SWS_DM_00031]	Starting time-based event debouncing for failed
[SWS_DM_00034]	Starting time-based event debouncing for passed
[SWS_DM_00037]	Debounce time freeze request
[SWS_DM_00042]	Canceling external service processors
[SWS_DM_00046]	Each `Diagnostic Conversation` has its own session resources
[SWS_DM_00047]	Each `Diagnostic Conversation` has its own security-level resources
[SWS_DM_00049]	Refusal of diagnostic request due to prioritization with BusyRepeatRequest
[SWS_DM_00061]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCByStatusMask
[SWS_DM_00062]	Mapping between ISO 14229-1[1] and Autosar Diagnostic Extract Template [2] of the DTCFormatIdentifier
[SWS_DM_00063]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord
[SWS_DM_00067]	Monitor initialization for clearing reason
[SWS_DM_00068]	Monitor initialization for operation cycle restart reason
[SWS_DM_00069]	Monitor initialization for enable condition re-enabling reason
[SWS_DM_00070]	Monitor initialization for `DTC` setting re-enabling reason
[SWS_DM_00071]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00074]	Handling of enable conditions
[SWS_DM_00089]	Reporting `kPrepassed` or `kPrefailed` for events without an assigned debouncing algorithm
[SWS_DM_00090]	Support of UDS service ClearDiagnosticInformation
[SWS_DM_00091]	Evaluation of ClearDiagnosticInformation parameters
[SWS_DM_00092]	Parameter range check for groupOfDTC request parameter
[SWS_DM_00096]	Validation Steps and Order
[SWS_DM_00098]	UDS message checks

▽

△

Number	Heading
[SWS_DM_00099]	Supported Service SID level checks
[SWS_DM_00100]	Supported Service subfunction level checks
[SWS_DM_00101]	Session Access SID level Permission
[SWS_DM_00102]	Session Access subfunction level Permission
[SWS_DM_00103]	Security Access level Permission
[SWS_DM_00104]	Supported UDS Services
[SWS_DM_00106]	Signature of Manufacturer Permission Check Method
[SWS_DM_00108]	Signature of Supplier Permission Check Method
[SWS_DM_00111]	Configurable environment condition checks
[SWS_DM_00112]	Condition check definition
[SWS_DM_00113]	Positive response for UDS service 0x14
[SWS_DM_00114]	Limitation to one simultaneous DTC clear operation
[SWS_DM_00115]	Memory error handling while clearing DTCs
[SWS_DM_00117]	Clearing a DTC
[SWS_DM_00121]	Forbidden clearing of snapshot records and extended data records
[SWS_DM_00122]	UDS response behavior on not allowed clear operations
[SWS_DM_00123]	Block status byte clearing during a clear DTC operation
[SWS_DM_00124]	Limited status byte clearing during a clear DTC operation
[SWS_DM_00126]	Realisation of UDS service 0x3E TesterPresent
[SWS_DM_00127]	Availability of diagnostic service processors
[SWS_DM_00128]	Realization of UDS service RequestDownload (0x34)
[SWS_DM_00129]	Supported addressAndLengthFormatIdentifier
[SWS_DM_00130]	Not supported addressAndLengthFormatIdentifier
[SWS_DM_00131]	UDS service RequestDownload (0x34) processing
[SWS_DM_00134]	Realization of UDS service RequestUpload (0x35)
[SWS_DM_00136]	UDS service RequestUpload (0x35) processing
[SWS_DM_00137]	Realization of UDS service TransferData (0x36)
[SWS_DM_00138]	UDS service TransferData (0x36) processing
[SWS_DM_00139]	UDS service TransferData (0x36) validation
[SWS_DM_00140]	Realisation of UDS service 0x28 CommunicationControl
[SWS_DM_00141]	Realization of UDS service RequestTransferExit (0x37)
[SWS_DM_00142]	UDS service RequestTransferExit (0x37) processing
[SWS_DM_00143]	UDS service RequestTransferExit (0x37) validation
[SWS_DM_00153]	Triggering for snapshot record storage
[SWS_DM_00156]	Triggering for extended data record storage and updates
[SWS_DM_00159]	Allow only to clear `GroupOfAllDTCs`

▽

△

Number	Heading
[SWS_DM_00160]	Allow to clear single DTCs
[SWS_DM_00162]	Point in time for positive response for ClearDTC
[SWS_DM_00163]	Definition of a failed clear operation with event clear allowed and event combination
[SWS_DM_00164]	Definition of a failed clear operation with event clear allowed and clearing a group of DTCs
[SWS_DM_00167]	Ignoring reported events for not started operation cycles
[SWS_DM_00168]	Availability of DiagnosticMonitor service interfaces
[SWS_DM_00169]	Restart of operation cycles
[SWS_DM_00170]	Realisation of UDS service ReadDataByIdentifier (0x22)
[SWS_DM_00177]	Reaction on ApplicationError
[SWS_DM_00186]	Realisation of UDS service WriteDataByIdentifier (0x2E)
[SWS_DM_00192]	Operation cycles are only ended once
[SWS_DM_00193]	Support of a user-defined fault memory clear request
[SWS_DM_00194]	Definition of the user-defined fault memory number for ClearDiagnosticInformation
[SWS_DM_00195]	Clearing a user-defined memory
[SWS_DM_00197]	Communication control service processing
[SWS_DM_00198]	Negative Response processing
[SWS_DM_00199]	Positive Response processing
[SWS_DM_00201]	Realization of UDS service RoutineControl (0x31)
[SWS_DM_00202]	Check for Supported RoutineIdentifier and Reaction
[SWS_DM_00203]	Check for Supported Subfunction and Reaction
[SWS_DM_00205]	Providing the VIN in DoIP protocol messages
[SWS_DM_00208]	Validation of the requested user-defined memory number
[SWS_DM_00210]	UDS Service RoutineControl (0x31) startRoutine processing
[SWS_DM_00211]	UDS Service RoutineControl (0x31) requestRoutineResults processing
[SWS_DM_00212]	UDS Service RoutineControl (0x31) stopRoutine processing
[SWS_DM_00213]	DTC status processing
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00216]	DTC status bit transitions triggered by operation cycle changes
[SWS_DM_00217]	DTC status bit transitions triggered by ClearDiagnosticInformation UDS service
[SWS_DM_00218]	Confirmation
[SWS_DM_00219]	Observability of the status byte
[SWS_DM_00220]	Notification about DTC status changes
[SWS_DM_00222]	Observability of indicator status

▽

△

Number	Heading
[SWS_DM_00226]	Support of UDS service DiagnosticSessionControl
[SWS_DM_00227]	Check for supported sessions
[SWS_DM_00228]	Switch to requested Diagnostic Session
[SWS_DM_00229]	Support of UDS service ControlDTCSetting (0x85)
[SWS_DM_00230]	Check for supported subfunctions
[SWS_DM_00231]	Invalid value for optional request parameter
[SWS_DM_00232]	Support of Subfunction 0x01 (ON)
[SWS_DM_00233]	Support of Subfunction 0x02 (OFF)
[SWS_DM_00234]	Support of UDS service ECUReset
[SWS_DM_00235]	ECUReset service processing
[SWS_DM_00236]	Realization of UDS service 0x27 SecurityAccess
[SWS_DM_00237]	Aging
[SWS_DM_00240]	Processing the aging counter
[SWS_DM_00241]	Aging cycle and threshold
[SWS_DM_00242]	Re-occurrence after aging
[SWS_DM_00243]	Aging-related UDS DTC status byte processing
[SWS_DM_00244]	Support of UDS service ReadDTCInformation, Subfunction 0x01
[SWS_DM_00245]	Support of UDS service ReadDTCInformation, Subfunction 0x02
[SWS_DM_00246]	Support of UDS service ReadDTCInformation, Subfunction 0x04
[SWS_DM_00247]	Support of UDS service ReadDTCInformation, Subfunction 0x07
[SWS_DM_00248]	Notification about session change
[SWS_DM_00249]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00250]	Notification about security-level change
[SWS_DM_00252]	Reaction on Unsupported Subfunction
[SWS_DM_00260]	instances of interface ClearDTC
[SWS_DM_00261]	Usage of ClearDTC Interface
[SWS_DM_00263]	ClearDTC call on invalid DTC or DTCgroup
[SWS_DM_00265]	ClearDTC called while another clear operation is in progress
[SWS_DM_00266]	ClearDTC processing in case of memory errors
[SWS_DM_00267]	Possible failure of ClearDTC
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00269]	Reaction on Unsupported Subfunction
[SWS_DM_00270]	Counting of attempts to change security level
[SWS_DM_00271]	Evaluate the number of failed security level change attempts
[SWS_DM_00273]	Notification event upon `snapshot record` updates
[SWS_DM_00277]	Cancellation of a Diagnostic Conversation in case of External Service Processing

▽

△

Number	Heading
[SWS_DM_00278]	Cancellation of a Diagnostic Conversation in case of Internal Processing
[SWS_DM_00279]	Cancellation of a Diagnostic Conversation before Response Transmission
[SWS_DM_00280]	Cancellation of a Diagnostic Conversation at Response Transmission
[SWS_DM_00286]	Configurable environmental condition check execution
[SWS_DM_00288]	Configurable environmental condition check evaluates to `TRUE`
[SWS_DM_00289]	Configurable environmental condition check evaluates to `FALSE`
[SWS_DM_00290]	Refusal of diagnostic request due to prioritization without response
[SWS_DM_00291]	
[SWS_DM_00293]	
[SWS_DM_00294]	
[SWS_DM_00296]	
[SWS_DM_00297]	
[SWS_DM_00298]	
[SWS_DM_00299]	
[SWS_DM_00300]	
[SWS_DM_00301]	
[SWS_DM_00302]	
[SWS_DM_00303]	
[SWS_DM_00304]	
[SWS_DM_00306]	
[SWS_DM_00307]	
[SWS_DM_00309]	
[SWS_DM_00310]	
[SWS_DM_00311]	
[SWS_DM_00312]	
[SWS_DM_00313]	
[SWS_DM_00314]	
[SWS_DM_00315]	
[SWS_DM_00319]	
[SWS_DM_00322]	
[SWS_DM_00323]	
[SWS_DM_00325]	
[SWS_DM_00326]	
[SWS_DM_00327]	
[SWS_DM_00329]	Lifecycle management of an `Uds Transport Protocol` implementation
[SWS_DM_00336]	
[SWS_DM_00337]	

▽

△

Number	Heading
[SWS_DM_00338]	
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00360]	EcuReset positive response processing after reset
[SWS_DM_00361]	EcuReset application error processing
[SWS_DM_00362]	Checking Supported Subfunction for CompareKey
[SWS_DM_00364]	Negative response processing
[SWS_DM_00365]	Suppression of positive response in accordance to ISO 14229-1[1]
[SWS_DM_00366]	Suppression of negative response for functional requests in accordance to ISO 14229-1[1]
[SWS_DM_00367]	No service processing
[SWS_DM_00368]	Sending busy responses
[SWS_DM_00369]	Maximum number of busy responses
[SWS_DM_00370]	Support of UDS service ReadDTCInformation, Subfunction 0x06
[SWS_DM_00371]	Support of UDS service ReadDTCInformation, Subfunction 0x14
[SWS_DM_00372]	Support of UDS service ReadDTCInformation, Subfunction 0x17
[SWS_DM_00373]	Support of UDS service ReadDTCInformation, Subfunction 0x18
[SWS_DM_00374]	Support of UDS service ReadDTCInformation, Subfunction 0x19
[SWS_DM_00376]	Positive response processing
[SWS_DM_00379]	Handling of storage conditions
[SWS_DM_00380]	Support for S3 timer
[SWS_DM_00381]	Session timeout
[SWS_DM_00384]	
[SWS_DM_00385]	Acceptance of UDS message reception
[SWS_DM_00386]	Ignoring UDS message reception because DM is busy
[SWS_DM_00393]	Retrieving data for `internal DiagnosticDataElement`s
[SWS_DM_00397]	Retrieving data for `external DiagnosticDataElement`s
[SWS_DM_00401]	Reading Diagnostic Data Identifier on Data Element level
[SWS_DM_00404]	Default Service Interface for reading `DiagnosticDataIdentifier`
[SWS_DM_00407]	Default Service Interface for writing `DiagnosticDataIdentifier`
[SWS_DM_00408]	Retrieving data for requested DataIdentifier
[SWS_DM_00412]	Check requested number of DataIdentifiers
[SWS_DM_00413]	Check supported DataIdentifier in active session
[SWS_DM_00414]	Check supported DataIdentifier on active security level
[SWS_DM_00416]	Check supported DataIdentifier in active session
[SWS_DM_00417]	Check supported DataIdentifier on active security level
[SWS_DM_00418]	Writing data for requested DataIdentifier
[SWS_DM_00419]	Reaction on ApplicationError

▽

△

Number	Heading
[SWS_DM_00420]	Instantiation of Diagnostic Server
[SWS_DM_00434]	Providing the `PowerMode` in DoIP protocol messages

Table B.8: Changed Traceables in 18-10

B.3.3 Deleted Traceables in 18-10

Number	Heading
[SWS_DM_00001]	SRS Diagnostics
[SWS_DM_00012]	DoIP configurable source address identification
[SWS_DM_00041]	Behavior according to ISO Multiple client handling flow
[SWS_DM_00043]	Request refusal in case of no resources
[SWS_DM_00044]	Request refusal in case of non-default session active
[SWS_DM_00045]	Ignore ISO same resource access check
[SWS_DM_00048]	Request refusal in case of no resources
[SWS_DM_00051]	Cancellation of `Active Protocol` with lower priority
[SWS_DM_00052]	Selection between multiple cancellation candidates
[SWS_DM_00066]	Monitor initialization
[SWS_DM_00105]	Configurable Manufacturer Permission Check Services
[SWS_DM_00107]	Configurable Supplier Permission Check Services
[SWS_DM_00118]	Event specific configuration to allow clearing of a DTC
[SWS_DM_00119]	Init value for events with clear allowed information
[SWS_DM_00120]	Description of application interface to control the clear event behavior
[SWS_DM_00125]	Linking between event clear allowed and clearing a DTC
[SWS_DM_00161]	Negative response on not supported GroupOfDTC parameter
[SWS_DM_00166]	Trigger to process event status
[SWS_DM_00180]	Provide Protocol Priority Configurability
[SWS_DM_00182]	Identification of a protocol for Priority Assignment
[SWS_DM_00183]	Wildcards per attribute
[SWS_DM_00184]	Protocol Match Search
[SWS_DM_00185]	No Match
[SWS_DM_00258]	Cancellation of `Active Protocol` in non-default session
[SWS_DM_00259]	Completion of already `Active Protocol`s in default session
[SWS_DM_00274]	Definition of an active diagnostic protocol
[SWS_DM_00281]	Cancellation of active DiagnosticConversation in Non-Default Session
[SWS_DM_00282]	Handling of non-/active diagnostic conversations
[SWS_DM_00295]	meta info map vendor type

▽

△

Number	Heading
[SWS_DM_00305]	Const UdsMessage Pointer vendor type
[SWS_DM_00308]	Global Channel Identifier type
[SWS_DM_00316]	Header file
[SWS_DM_00317]	UdsTransportProtocolHandler constructor
[SWS_DM_00318]	UdsTransportProtocolHandler destructor
[SWS_DM_00320]	UdsTransportProtocolHandler UdsTransportProtocolMgr member
[SWS_DM_00321]	constructor member initialization
[SWS_DM_00324]	UdsTransportProtocolHandler UdsTransportProtocolHandlerID member
[SWS_DM_00328]	UdsMessage Pointer vendor type
[SWS_DM_00334]	UdsTransportProtocolMgr may be an abstract class
[SWS_DM_00335]	Header file
[SWS_DM_00339]	ByteVector vendor type
[SWS_DM_00402]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00403]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00405]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00406]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00410]	Check session permission
[SWS_DM_00411]	Check security level permission
[SWS_DM_CONSTR_00267]	Required VINDataIdentifier

Table B.9: Deleted Traceables in 18-10

B.3.4 Added Constraints in 18-10

none

B.3.5 Changed Constraints in 18-10

none

B.3.6 Deleted Constraints in 18-10

none