Document Title	Specification of Diagnostics for Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	723

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	17-10

Document Change History

Date	Release	Changed by	Description
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)

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).

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, 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.
FDC	Fault Detection Counter according to 14229-1[1]
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.
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:
Active Protocol	A `Diagnostic Protocol` that has at least one of: • Active Diagnostic Request. • Elevated Session Level. • Elevated Security Access.
Aging	Unlearning/deleting of a no longer failed event/DTC after a defined number of operation cycles from event memory.
Diagnostic Protocol	Diagnostic Protocol is an ISO-14229 term, describing the diagnostic conversation between a distinguishable diagnostic client and the diagnostic server
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.
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.
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.

Terms:	Description:
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 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 Log and Trace for Adaptive Platform
AUTOSAR_SWS_AdaptiveLogAndTrace

[7] Requirements on Diagnostic
AUTOSAR_SRS_Diagnostic

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

[9] Road vehicles – Unified diagnostic services (UDS) – Part 5: Unified diagnostic services on Internet Protocol implementation (UDSonIP)
http://www.iso.org

[10] Road vehicles – Diagnostic communication over Internet Protocol (DoIP) – Part 2: Network and transport layer requirements and services
http://www.iso.org

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [5, SWS General], which is also valid for DM.

Thus, the specification SWS BSW General shall be considered as additional and required specification for DM.

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 domain / 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 *DoIP payload types* are not supported by the DM:
  - 0x0001 Vehicle identification request message
  - 0x0002 Vehicle identification request message with EID
  - 0x0003 Vehicle identification request message with VIN
  - 0x0004 Vehicle announcement message/vehicle identification response message

  - 0x0007 Alive check request
  - 0x0008 Alive check response
  - 0x4001 DoIP entity status request
  - 0x4002 DoIP entity status response
  - 0x4003 Diagnostic power mode information request
  - 0x8002 Diagnostic message positive acknowledgement
  - 0x8003 Diagnostic message negative acknowledgement
- The following UDS services are not implemented by the DM:
  - 0x24 ReadScalingDataByIdentifier
  - 0x2A ReadDataByPeriodicIdentifier
  - 0x2C DynamicallyDefineDataIdentifier
  - 0x2F InputOutputControlByIdentifier
  - 0x38 RequestFileTransfer
  - 0x3D WriteMemoryByAddress
  - 0x83 AccessTimingParameter
  - 0x84 SecuredDataTransmission
  - 0x86 ResponseOnEvent
  - 0x87 LinkControl
- The following UDS services are only supported with the interface GenericUDSService:
  - 0x11 ECUReset
  - 0x28 CommunicationControl
- Sub-functions of UDS services are implemented according to ISO 14229-1[1]. Unless this document is not saying otherwise, the DM implements the behavior of a UDS service according to ISO 14229-1[1].
- 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 (reportMirrorMemoryDTCExtDataRecordByDTCNumber)
  - 0x19 with subfunction 0x11 (reportNumberOfMirrorMemoryDTCByStatusMask)

- 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)
- Security Access: "Delay on boot" mechanism is not supported.
- Event Memory: Variant handling at runtime for events/DTCs is not supported.
- Event Memory: User controlled warning indicator bit 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

DM is a service and therefore uses ara::com to communicate with applications. The DM uses ara::log ([6], Log and Trace) for logging and tracing purposes. DM may use ara::per (Persistency) to store non-volatile data.

6 Requirements Tracing

The following tables reference the requirements specified in [7] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_Diag_04010]	Ensure interaction in order to fulfill ISO 14229-1 and ISO 15031-5	[SWS_DM_00058] [SWS_DM_00060] [SWS_DM_00061] [SWS_DM_00062] [SWS_DM_00063] [SWS_DM_00064] [SWS_DM_00065] [SWS_DM_00213] [SWS_DM_00214] [SWS_DM_00215] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00223] [SWS_DM_00237] [SWS_DM_00238] [SWS_DM_00239] [SWS_DM_00240] [SWS_DM_00241] [SWS_DM_00242] [SWS_DM_00243] [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] [SWS_DM_CONSTR_00082]
[SRS_Diag_04016]	Support ”Busy handling” by sending a negative response 0x78	[SWS_DM_00368] [SWS_DM_00369]
[SRS_Diag_04019]	Confirm transmitting if complete to continue processing	[SWS_DM_00341]
[SRS_Diag_04020]	Suppress responses to diagnostic tool requests	[SWS_DM_00365] [SWS_DM_00366]
[SRS_Diag_04150]	Support the primary fault memory defined by ISO 14229-1	[SWS_DM_00055] [SWS_DM_00056] [SWS_DM_00057] [SWS_DM_00083]
[SRS_Diag_04166]	Several tester conversations in parallel with assigned priorities	[SWS_DM_00011] [SWS_DM_00012] [SWS_DM_00016] [SWS_DM_00041] [SWS_DM_00042] [SWS_DM_00043] [SWS_DM_00044] [SWS_DM_00045] [SWS_DM_00046] [SWS_DM_00047] [SWS_DM_00048] [SWS_DM_00049] [SWS_DM_00051] [SWS_DM_00052] [SWS_DM_00180] [SWS_DM_00182] [SWS_DM_00183] [SWS_DM_00184] [SWS_DM_00185] [SWS_DM_00258] [SWS_DM_00259] [SWS_DM_00277] [SWS_DM_00278] [SWS_DM_00279] [SWS_DM_00280] [SWS_DM_00281] [SWS_DM_00282] [SWS_DM_00290]
[SRS_Diag_04167]	Conversation preemption/abortion	[SWS_DM_00042] [SWS_DM_00049] [SWS_DM_00051] [SWS_DM_00052] [SWS_DM_00180] [SWS_DM_00182] [SWS_DM_00183] [SWS_DM_00184] [SWS_DM_00185] [SWS_DM_00277] [SWS_DM_00278] [SWS_DM_00279] [SWS_DM_00280] [SWS_DM_00281] [SWS_DM_00282] [SWS_DM_00290]
[SRS_Diag_04168]	Adding of user-defined transport layers	[SWS_DM_00005]
[SRS_Diag_04169]	Provide an interface for external UDS service processors.	[SWS_DM_00197]
[SRS_Diag_04178]	Support operation cycles according to ISO 14229-1	[SWS_DM_00002] [SWS_DM_00003] [SWS_DM_00004] [SWS_DM_00169] [SWS_DM_00192] [SWS_DM_00216]

Requirement	Description	Satisfied by
[SRS_Diag_04179]	Provide interfaces for diagnostic monitors.	[SWS_DM_00007] [SWS_DM_00008] [SWS_DM_00166] [SWS_DM_00168]
[SRS_Diag_04180]	Process all UDS Services related to diagnostic fault memory of ISO 14229-1 internally	[SWS_DM_00090] [SWS_DM_00091] [SWS_DM_00092] [SWS_DM_00104] [SWS_DM_00116] [SWS_DM_00117] [SWS_DM_00144] [SWS_DM_00145] [SWS_DM_00146] [SWS_DM_00147] [SWS_DM_00159] [SWS_DM_00160] [SWS_DM_00161] [SWS_DM_00162] [SWS_DM_00163] [SWS_DM_00164] [SWS_DM_00165] [SWS_DM_00229] [SWS_DM_00230] [SWS_DM_00231] [SWS_DM_00232] [SWS_DM_00233]
[SRS_Diag_04183]	Notify interested parties about event status changes	[SWS_DM_00219] [SWS_DM_00220]
[SRS_Diag_04184]	No description	[SWS_DM_00273]
[SRS_Diag_04185]	Notify applications about the clearing of an event	[SWS_DM_00066] [SWS_DM_00067]
[SRS_Diag_04186]	Notify applications about the start or restart of an operation cycle	[SWS_DM_00066] [SWS_DM_00068]
[SRS_Diag_04188]	No description	[SWS_DM_00013] [SWS_DM_00014] [SWS_DM_00015] [SWS_DM_00017] [SWS_DM_00018] [SWS_DM_00019] [SWS_DM_00020] [SWS_DM_00021] [SWS_DM_00022] [SWS_DM_00023] [SWS_DM_00024] [SWS_DM_00025] [SWS_DM_00026] [SWS_DM_00028] [SWS_DM_00029] [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_00087] [SWS_DM_00088] [SWS_DM_00089]
[SRS_Diag_04191]	No description	[SWS_DM_00118] [SWS_DM_00119] [SWS_DM_00120] [SWS_DM_00121] [SWS_DM_00122] [SWS_DM_00123] [SWS_DM_00124] [SWS_DM_00125]
[SRS_Diag_04192]	Provide the ability to handle event specific enable and storage conditions	[SWS_DM_00072] [SWS_DM_00074] [SWS_DM_00078] [SWS_DM_00079]
[SRS_Diag_04194]	ClearDTC shall be accessible for applications	[SWS_DM_00260] [SWS_DM_00261] [SWS_DM_00262] [SWS_DM_00263] [SWS_DM_00264] [SWS_DM_00265] [SWS_DM_00266] [SWS_DM_00267]

Requirement	Description	Satisfied by
[SRS_Diag_04196]	UDS Service handling for all diagnostic services defined in ISO 14229-2	[SWS_DM_00104] [SWS_DM_00126] [SWS_DM_00127] [SWS_DM_00128] [SWS_DM_00129] [SWS_DM_00130] [SWS_DM_00131] [SWS_DM_00134] [SWS_DM_00136] [SWS_DM_00137] [SWS_DM_00138] [SWS_DM_00139] [SWS_DM_00140] [SWS_DM_00141] [SWS_DM_00142] [SWS_DM_00143] [SWS_DM_00170] [SWS_DM_00172] [SWS_DM_00173] [SWS_DM_00174] [SWS_DM_00175] [SWS_DM_00176] [SWS_DM_00177] [SWS_DM_00178] [SWS_DM_00179] [SWS_DM_00186] [SWS_DM_00188] [SWS_DM_00189] [SWS_DM_00190] [SWS_DM_00191] [SWS_DM_00198] [SWS_DM_00199] [SWS_DM_00201] [SWS_DM_00202] [SWS_DM_00203] [SWS_DM_00210] [SWS_DM_00211] [SWS_DM_00212] [SWS_DM_00234] [SWS_DM_00235] [SWS_DM_00236] [SWS_DM_00249] [SWS_DM_00268] [SWS_DM_00269] [SWS_DM_00274] [SWS_DM_00360] [SWS_DM_00361] [SWS_DM_00362] [SWS_DM_00363] [SWS_DM_00363] [SWS_DM_00364] [SWS_DM_00367]
[SRS_Diag_04197]	Clearing the user defined fault memory	[SWS_DM_00193] [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_00227] [SWS_DM_00228] [SWS_DM_00248] [SWS_DM_00250]
[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_00105] [SWS_DM_00106] [SWS_DM_00107] [SWS_DM_00108] [SWS_DM_00286] [SWS_DM_00287] [SWS_DM_00288] [SWS_DM_00289]
[SRS_Diag_04203]	Common checks on all supported UDS Services Requests	[SWS_DM_00098] [SWS_DM_00099] [SWS_DM_00100] [SWS_DM_00101] [SWS_DM_00102] [SWS_DM_00103] [SWS_DM_00111] [SWS_DM_00112] [SWS_DM_00252]
[SRS_Diag_04204]	Provide the current status of each warning indicator.	[SWS_DM_00221] [SWS_DM_00222] [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_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]

7 Functional specification

The DM implements the two main building blocks of diagnostics: event memory management and diagnostic service handling. Technically both are distinct things handled in independent chapters.

7.1 Diagnostic service management

7.1.1 Overview

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.
- 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.

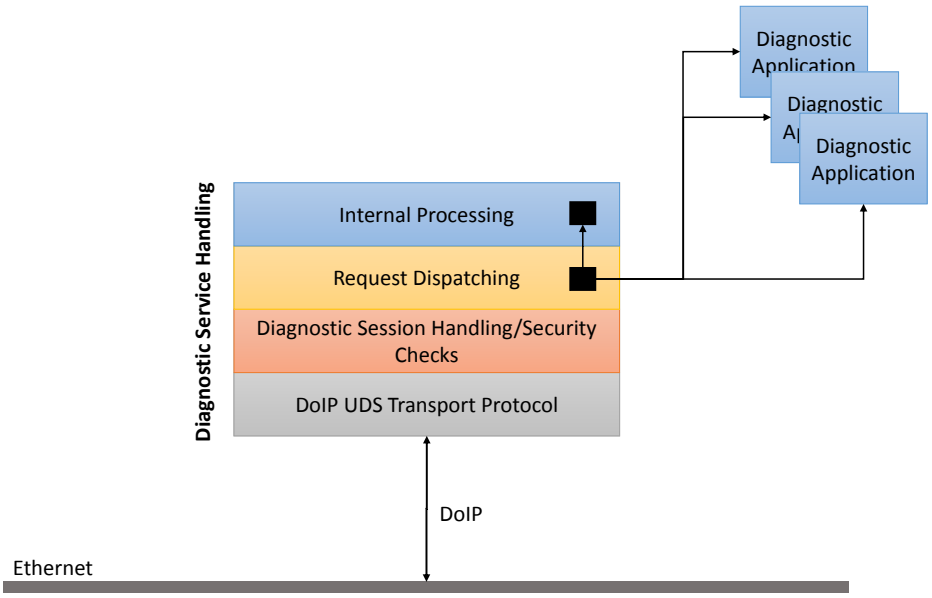

Figure 7.1: Architecture Diagnostic Service Handling

7.1.2 UDS Transport Layer

Currently the Adaptive Platform only supports Ethernet-based network technologies, which mandates support of DoIP[4]. It is very likely, that upcoming releases of the DM will also support CAN, CAN-FD, FR, ... networks. This is rather an architectural hint, to prepare enhancements of the DM. For future releases the DM will support various / different UDS Transport Layers beside DoIP.

7.1.2.1 DoIP

[SWS_DM_00005] DoIP Support ⌈ DM shall implement/provide an UDS Transport Layer implementation on Ethernet compliant with ISO-13400[4], also called DoIP. ⌋ *(SRS_Diag_04168)*

[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 any DoIP message, it shall obtain it by using the method Read of the service interface DataIdentifier with Diagnostic-DataIdentifier.representsVin set to true. ⌋*()*

Due to [SWS_DM_CONSTR_00207] there is always one unique DID defined to query the VIN and due to [SWS_DM_CONSTR_00206] there is a defined format for the VIN allowing the DM to identify and interpret the VIN data service in a matter to be compliant with ISO-13400[4].

7.1.2.2 Support of proprietary UDS Transport Layer

Since there exist OEM specific UDS Transport Layers, the DM supports a standardized C++ API, where custom/OEM specific UDS Transport Layers can be connected. This API is formally described in 8.3.1. Each proprietary `Uds Transport Protocol` implementation subclasses the abstract class `UdsTransportProtocolHandler`, which shall be provided by DM according to [SWS_DM_00315].

7.1.2.2.1 Initialization, Starting and Stopping of a proprietary UDS Transport-Layer

[SWS_DM_00329] Lifecycle management of an `Uds Transport Protocol` implementation ⌈ The lifecycle of an `Uds Transport Protocol` implementation, which respects the standardized API as described in 8.3.1, shall be managed by the DM in the following order:

- Creation of `Uds Transport Protocol` implementation by calling its constructor.
- 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])

⌋()

[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. ⌋()

[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. ⌋()

[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. ⌋()

[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`. ⌋*()*

[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. ⌋*()*

7.1.2.2.2 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

⌋*()*

[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. ⌋*()*

[SWS_DM_00343] Acceptance of UDS message reception ⌈ If the DM does not generally refuse the indicated request according to 7.1.3.3, it shall return a `UdsMessagePtr`, which owns a valid `UdsMessage` object. ⌋*()*

[SWS_DM_00344] Refusal of UDS message reception ⌈ If the DM does generally refuse the indicated request according to 7.1.3.3, it shall return a `UdsMessagePtr`, which does not own a valid `UdsMessage` object, but a nullptr instead. ⌋*()*

[SWS_DM_00345] Forwarding of UDS message ⌈ If the `Uds Transport Protocol` implementation has populated the payload of the returned `UdsMessagePtr` with the entire UDS request payload starting from `SID`, it shall call `HandleMessage` ([SWS_DM_00311]) on its `UdsTransportProtocolMgr` reference ((see [SWS_DM_00330]) with the returned `UdsMessagePtr` as argument. ⌋*()*

[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 `NotifyMessage-`

Failure ([SWS_DM_00310]) on its UdsTransportProtocolMgr reference ((see [SWS_DM_00330]) with the returned UdsMessagePtr as argument. ⌋()

7.1.2.2.3 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. ⌋()

[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]). ⌋()

[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]). ⌋()

[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. ⌋()

7.1.2.2.4 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. ⌋()

[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. ⌋*()*

[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. ⌋*()*

[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. ⌋*()*

7.1.3 Parallel Client Handling Variants

There are generally various approaches for a server (which the `DM` 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_00011] Selectability of parallelism concept ⌈ `DM` shall allow, that it can be configured, whether `DM` supports `fully parallel client concept` (7.1.3.5) or `pseudo parallel client concept` (7.1.3.4). ⌋*(SRS_Diag_04166, SRS_Diag_04209, SRS_Diag_04210)*

[SWS_DM_00016] Configurable number of supported parallel Diagnostic Clients ⌈ `DM` shall provide a configuration parameter, how many parallel Diagnostic Clients it shall support. This parameter is valid for both parallelism concepts. ⌋ *(SRS_Diag_04166)*

7.1.3.1 Definition of a Diagnostic Protocol

The parallelism in this context is based on the notion of a `Diagnostic Protocol`, which is a term introduced with ISO 14229-1[1]. A diagnostic protocol depicts a conversation between a distinct diagnostic client and the diagnostic server.

[SWS_DM_00274] Definition of an active diagnostic protocol ⌈ The `DM` shall consider a `diagnostic protocol` as active in the following cases:

- if a diagnostic request of a distinct Diagnostic Client is executed within **default session**, the `diagnostic protocol` is active from start of the diagnostic re-

quest (reception at DM) until DM has sent out the final positive or negative response.

- if a Diagnostic Client has entered a **non-default session**, the diagnostic protocol is active from start of the non-default session until this non-default session has ended. I.e. in non-default session a Diagnostic Protocol is active also across several diagnostic requests/responses.

⌋(*SRS_Diag_04196*)

[SWS_DM_00046] Each Diagnostic Protocol has own session resources ⌈ DM shall provide each Active Protocol with its own and independently managed diagnostic session, which can be any valid UDS session type. ⌋(*SRS_Diag_04166*)

[SWS_DM_00047] Each Diagnostic Protocol has own security-level resources ⌈ DM shall provide each Active Protocol with its own- and independently-managed security-level. ⌋(*SRS_Diag_04166*)

7.1.3.2 Identifying a Diagnostic Client

For the DM to identify to which Diagnostic Protocol a diagnostic request belongs, it has to exactly identify and distinguish between requests of different clients. 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[8]) 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 DM, 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_00012] DoIP configurable source address identification ⌈ The DoIP transport layer implementation shall support two configuration variants:

- Variant A: Only the source IP-address is used to identify the Diagnostic Client.
- Variant B: Source IP-address and port number are together used to identify the Diagnostic Client.

⌋(*SRS_Diag_04166*)

Note: Variant A is useful for a setup with exactly one tester software instance on the network node, which uses an arbitrary local port number on connect to the DM. In case this tester software sends a first request to the DM and then disconnects and reconnects to send the second request. During reconnect the tester software uses a different local port. In this case it is explicitly NOT intended that the port number is used to identify the Diagnostic Client, otherwise from the viewpoint of DM the 1st and the 2nd request would be assigned to different Diagnostic Client instances.

Opposite to this, variant B is useful for a setup where different logical tester software instances are located at the same network node, just differentiated by different local port numbers. In this setup it is necessary to use also the port number to identify the Diagnostic Client.

7.1.3.3 Refusing incoming Diagnostic request and Cancellation of Active Protocol

In the upcoming sections there are repeated requirements for the DM to refuse an incoming request or to cancel an Active Protocol. How DM shall accomplish this is generally described here:

[SWS_DM_00049] Refusal of second diagnostic request from different diagnostic client with BusyRepeatRequest ⌈ If a diagnostic request is already running and a second request from another diagnostic client can not be processed and the configuration parameter DiagnosticCommonProps.responseOnSecondDeclinedRequest is TRUE, the DM shall accept the second request and a negative response with NRC 0x21 (BusyRepeatRequest) shall be issued for the second request. ⌋(*SRS_Diag_04167, SRS_Diag_04166*)

[SWS_DM_00290] Refusal of second diagnostic request from different diagnostic client without response ⌈ If a diagnostic request is already running and a second request from another diagnostic client can not be processed and the configuration parameter DiagnosticCommonProps.responseOnSecondDeclinedRequest is FALSE, the DM shall accept and ignore the second request without a response. ⌋ (*SRS_Diag_04167, SRS_Diag_04166*)

[SWS_DM_00277] Cancellation of Active Protocol in case of External Service Processing ⌈ If DM decides to cancel an Active Protocol according to [SWS_DM_00051], in case a diagnostic request is currently processed on this protocol by a service processor external to DM, DM shall notify this external service processor, that the processing for this service shall be canceled according to [SWS_DM_00042]. ⌋(*SRS_Diag_04167, SRS_Diag_04166*)

[SWS_DM_00278] Cancellation of Active Protocol in case of Internal Processing ⌈ If DM decides to cancel an Active Protocol according to [SWS_DM_00051], in case a diagnostic request is currently processed on this protocol internally within DM, DM shall abort started activity as far as possible. ⌋(*SRS_Diag_04167, SRS_Diag_04166*)

[SWS_DM_00279] Cancellation of Active Protocol before Response Transmission ⌈ If DM decides to cancel an Active Protocol according to [SWS_DM_00051], in case a diagnostic request is currently processed on this protocol and response transmission has not yet been started, DM shall skip sending any response. ⌋ *(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00280] Cancellation of Active Protocol during Response Transmission ⌈ If DM decides to cancel an Active Protocol according to [SWS_DM_00051], in case a diagnostic request is currently processed on this protocol and diagnostic response transmission has already been started, DM shall gracefully abort transmission to the network layer. (What gracefully means depends on the UDS transport protocol in use for the canceled protocol. In DoIP, which uses TCP it typically means a TCP-ABORT/TCP-CLOSE) ⌋*(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00281] Cancellation of Active Protocol in Non-Default Session ⌈ If DM decides to cancel an Active Protocol according to [SWS_DM_00051], in case the protocol is in non-default session state, DM shall reset protocol to default-session state and update the field Status in DiagnosticProtocol. ⌋*(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00282] Handling of `CurrentActiveProtocols` ⌈ The DM shall remove a canceled diagnostic protocol from the field CurrentActiveProtocols of the service interface DiagnosticStatus. ⌋*(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00042] Canceling external service processors ⌈ External service processors, which are connected via service interfaces, which are described in **??** shall be canceled by call to Cancel() if the corresponding service interface provides this method. DM shall not wait for the return of the method (i.e. not wait for result of the ara::com::Future becoming available) as the Cancel() is only an asynchronous notification. ⌋*(SRS_Diag_04167, SRS_Diag_04166)*

7.1.3.4 Pseudo Parallel Concept

The characteristic of this parallelism concept 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 Protocols) get rejected with the exception if the new request maps to a protocol, which has a higher priority than the current Active Protocol. This characteristic of the 'pseudo parallel concept' means, that the diagnostic session state is not an individual state per diagnostic client (protocol), but it becomes a **global state for the entire DM** (and therefore typically the whole ECU).

[SWS_DM_00041] Behavior according to ISO Multiple client handling flow ⌈ In 'pseudo parallel concept' the DM shall follow the request handling flow of Figure J.2 in ISO 14229-1[1]. ⌋*(SRS_Diag_04166)*

[SWS_DM_00043] Request refusal in case of no resources ⌈ In 'pseudo parallel concept', if no Diagnostic Protocol is available, because the maximal configured number of parallel Active Protocols is already reached and no cancellation according to [SWS_DM_00051] applies, the DM shall refuse newly incoming request according to [SWS_DM_00049] and [SWS_DM_00290]. ⌋*(SRS_Diag_04166)*

[SWS_DM_00044] Request refusal in case of non-default session active ⌈ In 'pseudo parallel concept', if there is currently an Active Protocol, which has switched to a non-default session (and the entire DM is therefore in non-default session), the DM shall refuse newly incoming request from different client according to [SWS_DM_00049] and [SWS_DM_00290] except the new request belongs to a protocol with higher priority than the currently Active Protocol. ⌋*(SRS_Diag_04166)*

[SWS_DM_00258] Cancellation of Active Protocol in non-default session ⌈ In 'pseudo parallel concept', if there is currently an Active Protocol, which has switched to a non-default session and the new request belongs to a protocol with higher priority than the currently Active Protocol, the DM shall cancel the currently Active Protocol according to [SWS_DM_00277], [SWS_DM_00278], [SWS_DM_00279], [SWS_DM_00280], [SWS_DM_00281] and [SWS_DM_00282] and process the new request. ⌋*(SRS_Diag_04166)*

[SWS_DM_00259] Completion of already Active Protocols in default session ⌈ In 'pseudo parallel concept', if there is currently an Active Protocol in default session and another Active Protocol switches to non-default session concurrently, the Active Protocol in default session shall be regularly completed. ⌋ *(SRS_Diag_04166)*

Note: This means, that in this case also in 'pseudo parallel concept' there is a short timeframe until all Active Protocols in default session are completed, where the DM at the same time has Active Protocols in default and non-default session.

[SWS_DM_00045] Ignore ISO same resource access check ⌈ In 'pseudo parallel concept' the request handling flow of Figure J.2 in ISO 14229-1[1] requires a final check, whether the request to be executed will access the same resource as an already Active Protocol. This check shall be ignored by DM. ⌋*(SRS_Diag_04166)*

The DM can not identify, whether there is effectively a resource conflict, when two requests get processed in parallel, because there is no deduction from UDS request identification (SID, subfunction, options) to accessed machine resources! It is the job of the service implementation to care for resource management via locking mechanisms. If the service implementation detects an unresolvable resource conflict, it is able to report a NRC 0x21 on its own.

7.1.3.5 Fully Parallel Concept

The characteristic of this parallelism concept 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 Concept obviously requires more resources from the ECU (DM) acting as the server compared to the Pseudo Parallel Concept 7.1.3.4. 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 Concept is usually possible.

A DM configured for Fully Parallel Concept, allows, that it has at the same time N conversations (Active Protocols) with N different diagnostic clients, where each is in a — maybe different — non-default session.

[SWS_DM_00048] Request refusal in case of no resources ⌈ In 'fully parallel concept', if no Diagnostic Protocol is available, because maximum configured number of parallel Active Protocols is already reached and no cancellation according to [SWS_DM_00051] applies, the DM shall refuse the request according to [SWS_DM_00049] and [SWS_DM_00290]. ⌋*(SRS_Diag_04166)*

This requirement is basically identical to [SWS_DM_00043], but is intentionally repeated for the Fully Parallel Concept.

7.1.3.6 Protocol Prioritization and Cancellation

Both 'Parallel Client Handling Variants' depicted in 7.1.3.4 and 7.1.3.5 shall support the concept, that Diagnostic Protocols can be assigned a priority. If a new diagnostic request is received by the DM and DM discovers, that he has to assign a new protocol, but the number of allowed parallel Active Protocols (see [SWS_DM_00016]) has already been reached, prioritization could take place. That means: If the priority of the new incoming request is higher than any of the existing Active Protocols, the Active Protocol with the lowest priority will be cancelled and the new request processed.

[SWS_DM_00051] Cancellation of Active Protocol with lower priority ⌈ If DM detects, that he had to allocate a new protocol for an incoming request, but the configured maximum value of parallel Active Protocols has already been reached, DM shall cancel the Active Protocol with the lowest priority among the Active Protocols in case its priority is lower than the newly to-be-created protocol. ⌋ *(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00052] Selection between multiple cancellation candidates ⌈ If multiple Active Protocols with the same priority exist according to [SWS_DM_00051], DM shall prefer to cancel (see [SWS_DM_00277], [SWS_DM_00278], [SWS_DM_00279], [SWS_DM_00280], [SWS_DM_00281] and [SWS_DM_00282]) an Active Protocol, which is in default session over one in non-default session. ⌋*(SRS_Diag_04167, SRS_Diag_04166)*

Rationale: This requirement is only applicable in case of the Fully Parallel Concept. The idea behind this requirement is, that it is typically more costly to recover a diagnostic client conversation, which fell out from a non-default session, than to cancel an `Active Protocol` in default session, which comprises only one diagnostic request anyway.

7.1.3.7 Configurability of Protocol Priorities

[SWS_DM_00180] Provide Protocol Priority Configurability ⌈ `DM` shall assign the protocol a priority as defined in `DiagnosticProtocol.priority`. ⌋ *(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00182] Identification of a protocol for Priority Assignment ⌈ The identification of a diagnostic protocol to which a priority shall be assigned, shall be done based on the following attributes:

- UDS Source Address (SA) of the client.
- UDS Target Address Type (TA_Type, functional/physical)
- Protocol Type according to `DiagnosticProtocolType` (see 8.1.1.7)
- Network Endpoint Source Address of the client (format depends on `DiagnosticProtocolType`)

⌋*(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00183] Wildcards per attribute ⌈ Each of the attributes in [SWS_DM_00182] can be assigned a wildcard in the priority assignment. A wildcard means, it matches any value of the attribute (see [SWS_DM_00182]). ⌋ *(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00184] Protocol Match Search ⌈ The protocol priority assignments shall be organized in an ordered list, which `DM` shall search for a protocol match in ascending order. A Diagnostic Protocol to be started in the context of an incoming diagnostic request by `DM` shall be assigned the priority attribute of the first row, that matches regarding the attributes in [SWS_DM_00182]. ⌋*(SRS_Diag_04167, SRS_Diag_04166)*

[SWS_DM_00185] No Match ⌈ If no entry in the list matches the Diagnostic Protocol to be started in the context of an incoming diagnostic request, the priority shall be set to the lowest priority (255). ⌋*(SRS_Diag_04167, SRS_Diag_04166)*

Example of a protocol priority list with wildcards:

UDS Source Address (SA)	UDS Target Address Type	Protocol Type	Network Source Endpoint	Priority
0x00F0	*	UDS_ON_IP	192.16.200.1:*	3
*	*	UDS_ON_IP	192.16.200.1:*	4
*	*	OBD_ON_CAN	*	0
0x00F0	*	UDS_ON_CAN	*	1
*	*	UDS_ON_CAN	*	2

7.1.4 Request Validation/Verification

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

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

- **Server is busy?** Decision according to the chosen parallel client handling concept (see 7.1.3)
- **manufacturer specific failure detected?** Decision by applying manufacturer specific checks according to 7.1.4.4
- **SID supported?** Decision according to 7.1.4.2
- **SID supported in active session?** Decision according to 7.1.4.3
- **SID security check o.k.?** Decision according to 7.1.4.3
- **supplier-specific failure detected?** Decision by applying supplier-specific checks according to 7.1.4.4

[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. ⌋*()*

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

7.1.4.1 UDS request format checks

[SWS_DM_00098] UDS message checks ⌈ DM 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)*

7.1.4.2 Supported service checks

[SWS_DM_00099] Supported Service SID level checks ⌈ DM 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 ⌈ DM 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)*

7.1.4.3 Session and Security Checks

[SWS_DM_00101] Session Access SID level Permission ⌈ DM 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)*

[SWS_DM_00102] Session Access subfunction level Permission ⌈ DM 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)*

[SWS_DM_00103] Security Access level Permission ⌈ DM 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)*

7.1.4.4 Manufacturer and Supplier Permission Checks and Confirmation

[SWS_DM_00105] Configurable Manufacturer Permission Check Services ⌈ DM shall support a configurable ordered list of instances providing the service interface ServiceValidation (see 8.2.1.3), which are called to check whether the current service is accepted from manufacturer viewpoint. ⌋*(SRS_Diag_04199)*

[SWS_DM_00106] Signature of Manufacturer Permission Check Method ⌈ DM shall call the method Validate on service instances in ascending order of the configured manufacturer permission check service list. In case a call returned an ApplicationError of type UDSServiceFailed, the Verification shall be considered as failed and the negative response code shall be equal to the value of the errorContext provided by UDSServiceFailed. Any further calls of service instances of the configured list shall be aborted in this case. ⌋*(SRS_Diag_04199)*

[SWS_DM_00107] Configurable Supplier Permission Check Services ⌈ DM shall support a configurable ordered list of instances providing the service interface ServiceValidation (see 8.2.1.3), which are called to check whether the current service is accepted from supplier viewpoint. ⌋*(SRS_Diag_04199)*

[SWS_DM_00108] Signature of Supplier Permission Check Method ⌈ DM shall call the method Validate on service instances in ascending order of the supplier permission check service configured list. In case a call returned an ApplicationError of type UDSServiceFailed, the Verification shall be considered as failed and the negative response code shall be equal to the value of the errorContext provided by UDSServiceFailed. Any further calls of service instances of the configured list shall be aborted in this case. ⌋*(SRS_Diag_04199)*

[SWS_DM_00341] Confirmation of service processing ⌈ DM shall call the method Confirmation on service instances in ascending order of the configured manufacturer and afterwards supplier notification list in the following cases:

- UDS Transportlayer notified DM with TransmitConfirmation [SWS_DM_00312] about the final outcome of a transmit call
- Processing of diagnostic request finished and positive answer suppressed based on suppressPosRspMsgIndicationBit in the request

⌋*(SRS_Diag_04019)*

7.1.4.5 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 DM shall perform an 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 D-

DM shall send the NRC 0x22 (ConditionsNotCorrect) if the condition is not fulfilled. ⌋ *(SRS_Diag_04203)*

[SWS_DM_00112] Condition check definition ⌈ The DM 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_04203)*

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

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

7.1.5 Assemble positive or negative response

7.1.5.1 Positive Response

[SWS_DM_00363] Positive response processing ⌈ If an external service processor did not raise an ApplicationError, the DM shall return a positive response. ⌋ *(SRS_Diag_04196)*

7.1.5.2 Negative Response

[SWS_DM_00364] Negative response processing ⌈ If one of the external service processors raised an ApplicationError of type UDSServiceFailed, the DM shall return a negative response with the value of the errorContext of the ApplicationError. For details see ISO 14229-1[1]; chapter 10.2. ⌋*(SRS_Diag_04196)*

7.1.5.3 Suppression of Response

[SWS_DM_00365] Suppression of response ⌈ In the case that the "suppressPosRspMsgIndicationBit" is set in the request, the DM shall suppress the positive response. ⌋*(SRS_Diag_04020)*

[SWS_DM_00366] Suppression of response for functional requests ⌈ If one of the external service processors raised an ApplicationError of type UDSServiceFailed with one of the following values: ServiceNotSupported, SubfunctionNotSupported, RequestOutOfRange, SubFunctionNotSupportedInActiveSession and ServiceNotSupported and the request is functional addressed, the ApplicationError shall be suppressed. ⌋*(SRS_Diag_04020)*

7.1.5.4 No Processing and no Response

[SWS_DM_00367] No service processing ⌈ If one of the external service processors raised an ApplicationError of type UDSServiceFailed and the value NoProcessingNoResponse, the DM shall return no response and stop the service processing without further notification. ⌋*(SRS_Diag_04196)*

7.1.5.5 Sending busy Responses

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

[SWS_DM_00369] Max. 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 DM module shall stop processing the active diagnostic request, inform the application with the method Cancel of the interface DiagnosticProtocol and shall send a negative response with NRC 0x10 (General reject). ⌋*(SRS_Diag_04016)*

ï»¿

7.1.6 UDS service processing

This chapter describes the UDS service processing behavior of the DM.

[SWS_DM_00127] Availability of diagnostic service processors ⌈ The DM shall provide a service processor on SID level for all services by existence of a Di-

agnosticServiceClass referenced by a DiagnosticServiceInstance.serviceClass. ⌋(*SRS_Diag_04196*)

7.1.6.1 Supported UDS Services

[SWS_DM_00104] Supported UDS Services ⌈ DM shall support the following listed UDS services: ⌋(*SRS_Diag_04196, SRS_Diag_04180, SRS_Diag_04198*)

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

Note: Support Type Internally means, that the service with the given SID can be completely processed internally within DM module without relying on external functionality - typically in form of an AA. Support Type Externally means, that the DM 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.

7.1.6.2 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 DM and external service processors all addresses and data length information are mapped to a uint64 datatype.

[SWS_DM_00129] Supported addressAndLengthFormatIdentifier ⌈ The DM shall support for each nibble of the addressAndLengthFormatIdentifier a value between 1 and 8. ⌋(*SRS_Diag_04196*)

[SWS_DM_00130] Not supported addressAndLengthFormatIdentifier ⌈ The DM shall send the negative response 0x31 (requestOutOfRange), if an addressAndLengthFormatIdentifier with a value outside the range between 1 and 8 is received. ⌋ *(SRS_Diag_04196)*

7.1.6.3 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 DM 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 DM shall return a NRC 0x12 (SubfunctionNotSupported). ⌋*(SRS_Diag_04198)*

In the context of parallel clients, a DiagnosticSessionControl may lead to negative responses even for supported Subfunctions with positive permission checks, for details see Chapter 7.1.3.

[SWS_DM_00228] Switch to requested Diagnostic Session ⌈ On positive evaluation of a DiagnosticSessionControl request, the DM 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 DM did successfully change the session for a protocol, it shall update the field CurrentActiveProtocols of provided service DiagnosticStatus (see **??**) accordingly. ⌋ *(SRS_Diag_04198)*

7.1.6.4 Service 0x11 – ECUReset

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

[SWS_DM_00235] ECUReset service processing ⌈ The DM shall call the method Service of the interface GenericUDSService to process an ECU-Reset. ⌋ *(SRS_Diag_04196)*

[SWS_DM_00268] EcuReset positive response processing before reset ⌈ If the external processor did NOT raise an ApplicationError, the DM shall return a positive response before the actual reset, in case the parameter DiagnosticEcuReset-

Class.respondToReset is present and set to DiagnosticResponseToEcuResetEnum.respondBeforeReset. ⌋(*SRS_Diag_04196*)

[SWS_DM_00360] EcuReset positive response processing after reset ⌈ If the external processor did NOT raise an ApplicationError, the DM shall return a positive response after the actual reset if NotifyReestablishment method (see [SWS_DM_00326]) is called, 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 the external processor did raise an ApplicationError, the DM shall return immediately a negative response with the given NRC code in ApplicationError. ⌋(*SRS_Diag_04196*)

[SWS_DM_00269] Reaction on Unsupported Subfunction ⌈ The DM shall send a negative response 0x12 (SubfunctionNotSupported), if the requested subfunction value is neither in range of default subfunction values (requestType, see ISO 14229-1[1]) nor in range of the configured DiagnosticEcuReset.customSubFunctionNumber in the ECU. (see 7.1.4.2). ⌋(*SRS_Diag_04196*)

7.1.6.5 Service 0x14 – ClearDiagnosticInformation

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

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

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

[SWS_DM_00092] Parameter range check for groupOfDTC request parameter ⌈ The DM 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*)

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

The DTC clearing behavior is described in detail in chapter 7.2.4.6. 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 DM shall deny an UDS request 0x14 and send a negative response 0x22 (conditionsNotCorrect). ⌋*()*

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

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 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_00118], the DM 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_00118]
- it was requested to clear a DTC group and all the DTCs of the DTC group could not be cleared according to [SWS_DM_00118]
  (This doesn't apply when one or more DTC are allowed to be cleared.)

⌋*(SRS_Diag_04191)*

[SWS_DM_00159] Allow only to clear `GroupOfAllDTCs` ⌈ If the configuration `DiagnosticCommonProps.clearDtcLimitation` is set to `clearAllDtcs`, the DM 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 DM shall return a negative response 0x31 (RequestOutOfRange). ⌋ *(SRS_Diag_04180)*

[SWS_DM_00160] Allow to clear single `DTCs` ⌈ If the configuration `DiagnosticCommonProps.clearDtcLimitation` is set to `allSupportedDtcs`, the DM shall allow to clear single DTCs or DTCGroups. [SWS_DM_00092] defines the possible and refused values. ⌋*(SRS_Diag_04180)*

[SWS_DM_00161] Negative response on not supported GroupOfDTC parameter ⌈ If the DM is requested to clear a DTC or groupOfDTC different to GroupOfAllDTCs and the DM shall only clear GroupOfAllDTCs according to [SWS_DM_00148], the DM shall return a negative response 0x31 (RequestOutOfRange). ⌋*(SRS_Diag_04180)*

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

[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 `Diag-`

nosticEventToTroubleCodeUdsMapping referencing this DiagnosticEvent-ToTroubleCodeUdsMapping.troubleCodeUds the DM shall send a negative response 0x22 (conditionsNotCorrect) if one event forbids the clearance of the DTC according to [SWS_DM_00125]. ⌋(*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 DM 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_00125]. ⌋ (*SRS_Diag_04180*)

7.1.6.5.1 Clearing user-defined fault memory

According to [SWS_DM_00090] the DM 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 DM 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 DM 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 DM receives a an UDS service 0x14 ClearDiagnosticInformation with a length of 5 bytes, the DM 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 DM receives an 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. ⌋()

[SWS_DM_00195] Clearing a user-defined memory ⌈ If the DM 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 DM shall clear the requested user-defined fault memory. ⌋(*SRS_Diag_04197*)

For details about the fault memory clearing process please also refer to chapter 7.2.4.6.

[SWS_DM_00208] Validation of the requested user-defined memory number ⌈ If the DM 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 DM shall return a NRC 0x31 (RequestOutOfRange). ⌋ *(SRS_Diag_04197)*

7.1.6.6 Service 0x19 – ReadDTCInformation

Some UDS responses for the Service "0x19 – ReadDTCInformation" use the parameter "DTCFormatIdentifier" as part of the response PDU. The DM 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 DM shall derive the value from DiagnosticCommonProps.typeOfDtcSupported applying the following mapping rule: ⌋*(SRS_Diag_04010)*

typeOfDtcSupported	"DTCFormatIdentifier"
iso11992_4	0x03
iso14229_1	0x01
saeJ2012_da	0x00

7.1.6.6.1 SF 0x01 – reportNumberOfDTCByStatusMask

[SWS_DM_00244] Support of UDS service ReadDTCInformation, Subfunction 0x01 ⌈ The DM 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_04010)*

[SWS_DM_00061] Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCByStatusMask ⌈ While sending the positive response for ReadDTCInformation.reportNumberOfDTCByStatusMask, the DM shall set the response PDU "DTCFormatIdentifier" according to the mapping of [SWS_DM_00062]. ⌋*(SRS_Diag_04010)*

7.1.6.6.2 SF 0x02 – reportDTCByStatusMask

[SWS_DM_00245] Support of UDS service ReadDTCInformation, Subfunction 0x02 ⌈ The DM 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_04010*)

7.1.6.6.3 SF 0x04 – reportDTCSnapshotRecordByDTCNumber

[SWS_DM_00246] Support of UDS service ReadDTCInformation, Subfunction 0x04 ⌈ The DM 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_04010*)

7.1.6.6.4 SF 0x06 – reportDTCExtDataRecordByDTCNumber

[SWS_DM_00370] Support of UDS service ReadDTCInformation, Subfunction 0x06 ⌈ The DM 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_04010*)

7.1.6.6.5 SF 0x07 – reportNumberOfDTCBySeverityMaskRecord

[SWS_DM_00247] Support of UDS service ReadDTCInformation, Subfunction 0x07 ⌈ The DM 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_04010*)

[SWS_DM_00063] Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord ⌈ While sending the positive response for ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord, the DM shall set the response PDU "DTCFormatIdentifier" according to the mapping of [SWS_DM_00062]. ⌋ (*SRS_Diag_04010*)

7.1.6.6.6 SF 0x14 – reportDTCFaultDetectionCounter

[SWS_DM_00371] Support of UDS service ReadDTCInformation, Subfunction 0x14 ⌈ The DM 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_04010*)

7.1.6.6.7 SF 0x17 – reportUserDefMemoryDTCByStatusMask

[SWS_DM_00372] Support of UDS service ReadDTCInformation, Subfunction 0x17 ⌈ The DM 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_04010*)

7.1.6.6.8 SF 0x18 – reportUserDefMemoryDTCSnapshotRecordByDTCNumber

[SWS_DM_00373] Support of UDS service ReadDTCInformation, Subfunction 0x18 ⌈ The DM 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_04010*)

7.1.6.6.9 SF 0x19 – reportUserDefMemoryDTCExtDataRecordByDTCNumber

[SWS_DM_00374] Support of UDS service ReadDTCInformation, Subfunction 0x19 ⌈ The DM 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_04010*)

7.1.6.7 Service 0x22 – ReadDataByIdentifier

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

[SWS_DM_00178] Check requested number of DataIdentifiers ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), if the number of the requested DataIdentifiers exceeds the configured maximum number of DataIdentifiers (refer to configuration

parameter maxDidToRead), the DM shall return a negative response 0x13 (Incorrect message length or invalid format) ⌋(*SRS_Diag_04196*)

[SWS_DM_00172] Reaction on Unsupported DataIdentifier ⌈ The DM shall check, whether the DataIdentifier requested by the ReadDataByIdentifier is supported by an existing DiagnosticReadDataByIdentifier in the role dataIdentifier in the configuration. If the requested DataIdentifier is not supported by the configuration a negative response 0x31 (requestOutOfRange) shall be returned. ⌋(*SRS_Diag_04196*)

7.1.6.7.1 Internal DataIdentifiers

Internal DataIdentfiers are predefined DataIdentifiers of the ISO 14229-1[1] according to Table C.1, which can be fully provided by DM without a call to external AA. They are defined in this section.

[SWS_DM_00173] Classification as Internally implemented DID ⌈ A DataIdentifier shall be classified as internally implemented, if it is explicitly listed in this section and there is no DiagnosticServiceDataMapping or DiagnosticServiceSwMapping referencing a DiagnosticDataElement, which is aggregated by a DiagnosticParameter in the role dataIdentifier. ⌋(*SRS_Diag_04196*)

Rationale: If there is an explicit mapping in the form DiagnosticServiceDataMapping or DiagnosticServiceSwMapping to the DiagnosticReadDataByIdentifier, then the integration explicitly wants to have it implemented externally.

[SWS_DM_00174] Internally implemented DID ActiveDiagnosticSessionDataIdentifier ⌈ The DM shall implement the diagnostic service 0x22 ReadDataByIdentifier for DataIdentifier 0xF186 “ActiveDiagnosticSessionDataIdentifier” completely internally if classified “internal” according to [SWS_DM_00173], by returning the current active session of the protocol on which the request has been received. ⌋ (*SRS_Diag_04196*)

7.1.6.7.2 External DataIdentifiers

External DataIdentfiers are DataIdentifiers, which are provided by functionality outside the DM (typically in the form of an AA).

[SWS_DM_00175] Classification as Externally implemented DID ⌈ A DataIdentifier shall be classified as externally implemented, if there is a DiagnosticServiceDataMapping or DiagnosticServiceSwMapping referencing a DiagnosticDataElement, which is aggregated by a DiagnosticParameter in the role dataIdentifier. ⌋(*SRS_Diag_04196*)

[SWS_DM_00176] External ReadDataByIdentifier processing ⌈ The DM shall call the method Read of the interface DataIdentifier for every dataIdentifier (see 8.2.1.4). ⌋(*SRS_Diag_04196*)

[SWS_DM_00177] Negative Response processing ⌈ If one of the external processors raised an ApplicationError of type UDSServiceFailed for a specific dataIdentifier, the DM shall return a negative response with the value of the errorContext of the ApplicationError for the specific dataIdentifier. The negative response may be different for different dataIdentifier, for details see ISO 14229-1[1]; chapter 10.2. ⌋*(SRS_Diag_04196)*

[SWS_DM_00179] Positive Response processing ⌈ If an external processors did not raise an ApplicationError, the DM shall return a positive response for the specific dataIdentifier, even if one or more of the requested dataIdentifier have raised an ApplicationError of type UDSServiceFailed. If none of the external processors did raise an ApplicationError of type UDSServiceFailed, the DM shall return a positive response for all the dataIdentifiers. ⌋*(SRS_Diag_04196)*

7.1.6.8 Service 0x27 – SecurityAccess

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

[SWS_DM_00249] Checking Supported Subfunction for RequestSeed ⌈ The DM shall call GetSeed when the requested subfunction value (access type) is similar to the value of the instance of DiagnosticSecurityAccess with requestSeedId. ⌋ *(SRS_Diag_04196)*

[SWS_DM_00362] Checking Supported Subfunction for CompareKey ⌈ The DM 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_04196)*

[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 DM did successfully change the security-level for a protocol, it shall update the field CurrentActiveProtocols of provided service DiagnosticStatus (see **??**) accordingly. Whether a security level is applicable by the DiagnosticSecurityAccess is defined by securityLevel. ⌋*(SRS_Diag_04198)*

[SWS_DM_00270] Counting of attempts to change security level ⌈ The DM 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. ⌋*()*

[SWS_DM_00271] Evaluate the number of failed security level change attempts ⌈ The DM 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 DM module shall send a negative response with NRC 0x35 (InvalidKey).
If the number of failed attempts reaches the threshold value numFailedSecurityAccess the DM 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. ⌋()

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).
⌋()

7.1.6.9 Service 0x28 – CommunicationControl

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

[SWS_DM_00252] Reaction on Unsupported Subfunction ⌈ The DM 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 DM shall call the method Service of the interface GenericUDSService to process a communication control service. ⌋*(SRS_Diag_04169)*

[SWS_DM_00198] Negative Response processing ⌈ If at least one of the external processors raised an ApplicationError of type UDSServiceFailed, the DM shall return a negative response with the value of the errorContext of the ApplicationError. ⌋*(SRS_Diag_04196)*

[SWS_DM_00199] Positive Response processing ⌈ If none of the external processors did raise an ApplicationError, the DM shall return a positive response. ⌋*(SRS_Diag_04196)*

7.1.6.10 Service 0x2E – WriteDataByIdentifier

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

[SWS_DM_00188] Reaction on Unsupported DataIdentifier ⌈ The DM shall check, whether the DataIdentifier addressed by the WriteDataByIdentifier is supported by an existing DiagnosticWriteDataByIdentifier in the role dataIdentifier in the configuration. If the DataIdentifier addressed by the WriteDataByIdentifier is not supported in the configuration a negative response 0x31 (requestOutOfRange) shall be returned. ⌋*(SRS_Diag_04196)*

[SWS_DM_00189] WriteDataByIdentifier processing ⌈ The DM shall call the method method Write of the interface DataIdentifier (see 8.2.1.4). ⌋*(SRS_Diag_04196)*

[SWS_DM_00190] Negative Response processing ⌈ If the external processor raised an ApplicationError of type UDSServiceFailed, the DM shall return a negative response with the value of the errorContext of the ApplicationError. ⌋*(SRS_Diag_04196)*

[SWS_DM_00191] Positive Response processing ⌈ If the external processor did NOT raise an ApplicationError, the DM shall return a positive response. ⌋*(SRS_Diag_04196)*

7.1.6.11 Service 0x31 – RoutineControl

[SWS_DM_00201] Realisation of UDS service 0x31 RoutineControl ⌈ The DM shall implement the diagnostic service 0x31 RoutineControl 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 DM shall check, whether the RoutineIdentifier addressed by the RoutineControl is supported by an existing DiagnosticRoutine with a matching id in the configuration. If the RoutineIdentifier addressed by the RoutineControl is not supported a negative response with NRC 0x31 (requestOutOfRange) shall be returned. ⌋*(SRS_Diag_04196)*

[SWS_DM_00203] Check for Supported Subfunction and Reaction ⌈ The DM shall check, whether the Subfunction addressed by the RoutineControl 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 RoutineControl is not supported by the configuration a negative response NRC 0x12 (SubfunctionNotSupported) shall be returned. ⌋*(SRS_Diag_04196)*

[SWS_DM_00210] RoutineControl startRoutine processing ⌈ The DM shall call the method Start of the interface RoutineService (see 8.2.1.5) to process the subfunction startRoutine. ⌋*(SRS_Diag_04196)*

[SWS_DM_00211] RoutineControl requestRoutineResults processing ⌈ The DM shall call the method RequestResults of the interface RoutineService (see 8.2.1.5) to process the subfunction requestRoutineResults. ⌋*(SRS_Diag_04196)*

[SWS_DM_00212] RoutineControl stopRoutine processing ⌈ The DM shall call the method Stop of the interface RoutineService (see 8.2.1.5) to process the subfunction stopRoutine. ⌋*(SRS_Diag_04196)*

7.1.6.12 Service 0x34 – RequestDownload

[SWS_DM_00128] Realisation of UDS service 0x34 RequestDownload ⌈ The DM shall implement the diagnostic service 0x34 RequestDownload according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00131] Request download service processing ⌈ The DM shall call the method Service of the interface GenericUDSService to process a request download service. ⌋*(SRS_Diag_04196)*

7.1.6.13 Service 0x35 – RequestUpload

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

[SWS_DM_00136] Request upload service processing ⌈ The DM shall call the method Service of the interface GenericUDSService to process a request upload service. ⌋*(SRS_Diag_04196)*

7.1.6.14 Service 0x36 – TransferData

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

[SWS_DM_00138] Transfer data service processing ⌈ The DM shall call the method Service of the interface GenericUDSService to process a transfer data service. ⌋*(SRS_Diag_04196)*

ISO 14229-1[1] provides a service 0x36 specific NRC evaluation sequence. This sequence has checks that in rotating order needs to be done by the DM 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 service 0x36.

[SWS_DM_00139] Transfer data service validation ⌈ The DM shall realize all service specific NRC validation with the GenericUDSService of the service processors. ⌋ *(SRS_Diag_04196)*

7.1.6.15 Service 0x37 – RequestTransferExit

[SWS_DM_00141] Realisation of UDS service 0x37 RequestTransferExit ⌈ The DM shall implement the diagnostic service 0x37 RequestTransferExit according to ISO 14229-1[1]. ⌋*(SRS_Diag_04196)*

[SWS_DM_00142] Transfer data service processing ⌈ The DM shall call the method Service of the interface GenericUDSService to process a transfer data service. ⌋ *(SRS_Diag_04196)*

[SWS_DM_00143] Transfer data service validation ⌈ The DM shall realize all service specific NRC validation with the GenericUDSService of the service processors. ⌋ *(SRS_Diag_04196)*

7.1.6.16 Service 0x3E – TesterPresent

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

7.1.6.17 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 ⌈ The DM shall provide the UDS service 0x85 ControlDTCSetting according to ISO 14229-1[1]. ⌋ *(SRS_Diag_04180)*

[SWS_DM_00230] Check for supported subfunctions ⌈ If the Subfunction addressed by the ControlDTCSetting 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 DM shall return a NRC 0x12 (SubfunctionNotSupported). ⌋*(SRS_Diag_04180)*

[SWS_DM_00231] Invalid value for optional request parameter ⌈ If the DM receives a ControlDTCSetting request with DTCSettingControlOptionRecord != 0xFFFFFF, the DM shall send a NRC 0x31 (RequestOutOfRange). ⌋*(SRS_Diag_04180)*

[SWS_DM_00232] Support of Subfunction 0x01 (ON) ⌈ The DM shall support ControlDTCSetting with subfunction 0x01 (ON). If the DM receives a ControlDTCSetting

with Subfunction 0x01 (ON) and optionally with DTCSettingControlOptionRecord of value 0xFFFFFF, the DM shall enable the storage of all events and UDS status byte updates. ⌋(*SRS_Diag_04180*)

[SWS_DM_00233] Support of Subfunction 0x02 (OFF) ⌈ The DM shall support ControlDTCSetting with subfunction 0x02 (OFF). If the DM receives a ControlDTCSetting with Subfunction 0x02 (OFF) and optionally with DTCSettingControlOptionRecord of value 0xFFFFFF, the DM shall disable the storage of all events and UDS status byte updates. ⌋(*SRS_Diag_04180*)

7.2 Event memory management

7.2.1 Diagnostic Events

7.2.1.1 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.

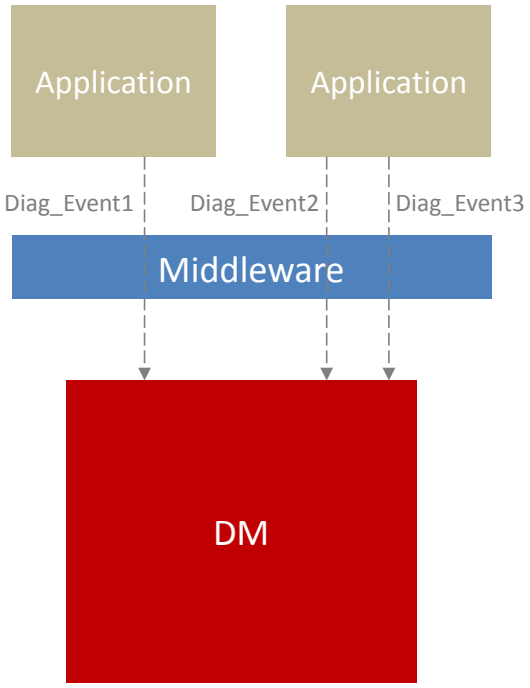

Figure 7.2: Example of diagnostic event usage

[SWS_DM_00007] Uniqueness of diagnostic events ⌈ An event is unique within the system and the DM shall only support notifications for events 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)*

7.2.1.2 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.

[SWS_DM_00066] Monitor initialization ⌈ The DM shall provide the InitMonitor event of the DiagnosticEventNotification service interface to trigger the initialization of diagnostic monitors. The event shall be of type InitMonitorReasonType and shall indicate the reason of initialization. ⌋*(SRS_Diag_04185, SRS_Diag_04186)*

[SWS_DM_00067] Monitor initialization for clearing reason ⌈ The DM shall publish the InitMonitor event with CLEAR value, in case the DTC mapped to the diagnostic event is cleared via the ClearDTC method of the ClearDTC service interface. ⌋ *(SRS_Diag_04185)*

[SWS_DM_00068] Monitor initialization for operation cycle restart reason ⌈ The DM shall publish the InitMonitor event with RESTART value, in case the operation cycle of the diagnostic event is (re)started by setting the OperationCycleState field of the OperationCycle service interface. ⌋*(SRS_Diag_04186)*

[SWS_DM_00069] Monitor initialization for enable condition reenabling reason ⌈ The DM shall publish the InitMonitor event with REENABLED value, in case an enable condition mapped to the diagnostic event is changed to fulfilled and this way all related enable conditions of the event are fulfilled. ⌋*()*

The detailed description of enable conditions can be found in 7.2.4.3 chapter.

[SWS_DM_00070] Monitor initialization for DTC setting reenabling reason ⌈ The DM shall publish the InitMonitor event with REENABLED value, in case DTC setting is reenabled via the UDS job ControlDTCSetting 0x85. ⌋*()*

[SWS_DM_00071] Monitor initialization for storage condition reenabling reason ⌈ The DM shall publish the InitMonitor event with STORAGEREENABLED value, if:

- a storage condition mapped to the diagnostic event is changed to fulfilled
- all related storage conditions of the event are fulfilled
- a FAILED or PASSED status was reported to the event while its storage conditions were disabled

⌋*()*

The detailed description of storage conditions can be found in 7.2.4.4 chapter.

7.2.1.3 Reporting

Diagnostic events are reported by applications via the method MonitorAction of service interface DiagnosticMonitor. The reported event status is processed by the DM, during the processing the event and 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 event in some situations.

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

[SWS_DM_00166] Trigger to process event status ⌈ If MonitorAction of service interface DiagnosticEvent is called, the DM shall process the reported diagnostic event status. ⌋*(SRS_Diag_04179)*

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

7.2.1.4 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_04188)*

Monitors will report a EventStatusByteType of PREPASSED or PREFAILED for events that are debounced by the DM.

[SWS_DM_00089] Reporting PREPASSED or PREFAILED for events without assigned debouncing algorithm ⌈ If MonitorAction is called with PREPASSED or PREFAILED for an event without assigned deboucing algorith, the DM shall interpret a reported PREPASSED as PASSED and PREFAILED as FAILED. ⌋*(SRS_Diag_04188)*

7.2.1.4.1 Debounce algorithm initialization

The DM provides debounce algorithms that are based on internal counters. This chapter describes the general applicable requirements for initialization and resetting these counters for all DM supported debounce algorithms.

[SWS_DM_00085] Debounce counter values after startup ⌈ Upon startup the DM shall set the debounce counters for all events with activated debouncing to 0. ⌋ *(SRS_Diag_04188)*

For how to activate the debouncing for a certain event refer to [SWS_DM_00014] and [SWS_DM_00015].

[SWS_DM_00086] Debounce counter value after clearing DTC ⌈ If the DM executes a ClearDTC command, the DM shall set all debounce counters to 0 for all events that have a DiagnosticEventToTroubleCodeUdsMapping to one of the cleared DTCs. ⌋*(SRS_Diag_04188)*

7.2.1.4.2 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 ⌈ The existence of a DiagnosticEventToDebounceAlgorithmMapping with an aggregated DiagEventDebounceCounterBased by DiagnosticDebounceAlgorithmProps.debounceAlgorithm shall activate a counter-based debouncing for this event. ⌋*(SRS_Diag_04188)*

[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_04188)*

[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_04188)*

[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_04188)*

[SWS_DM_00024] Qualified failed event using counter-based debouncing ⌈ If the internal debounce counter is greater or equal to DiagEventDebounceCoun-

terBased.counterFailedThreshold the DM shall process the event as FAILED. ⌋(*SRS_Diag_04188*)

[SWS_DM_00020] Internal debounce counter decrementation ⌈ The DM shall decrement the event's internal debounce counter by the configured step-size value of DiagEventDebounceCounterBased.counterDecrementStepSize, when the monitor reports PREPASSED. ⌋(*SRS_Diag_04188*)

[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_04188*)

[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_04188*)

[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_04188*)

[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_04188*)

[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_04188*)

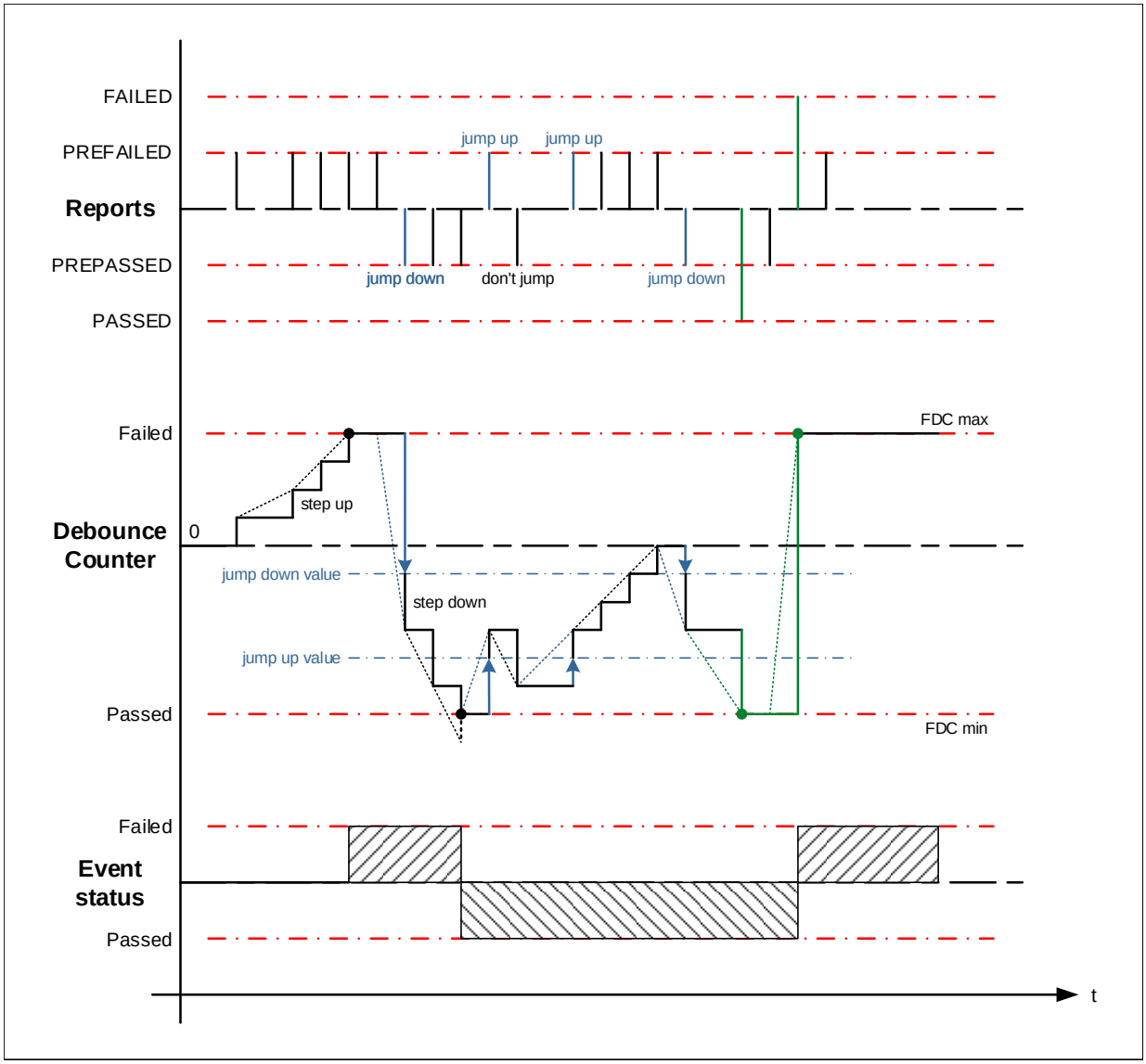

Figure 7.3: Counter-based debouncing

[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_04188)*

7.2.1.4.3 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 aggregated `Di-`

agEventDebounceTimeBased by DiagnosticDebounceAlgorithmProps.debounceAlgorithm shall activate a time-based debouncing for this event. ⌋ *(SRS_Diag_04188)*

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_04188)*

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] Point in time to start time-based event debouncing ⌈ The DM module shall start the debounce timer to qualify the reported event as failed when the monitor reports PREFAILED. ⌋*(SRS_Diag_04188)*

[SWS_DM_00032] Restrictions on restarting a running event debounce timer for failed ⌈ If the debounce timer of a specific event was already started to qualify the reported event as failed or the event is qualified as failed, the DM shall not restart the debounce timer upon a further report of PREFAILED. ⌋*(SRS_Diag_04188)*

[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_04188)*

[SWS_DM_00034] Starting the debounce timer ⌈ If the debounce timer to qualify the reported event as failed is not running for an event and PREFAILED is reported, the DM module shall start the debounce timer for this event. ⌋*(SRS_Diag_04188)*

[SWS_DM_00035] Restrictions on restarting a running event debounce timer for passed ⌈ If the debounce timer of a specific event was already started to qualify the reported event as passed or the event is qualified as passed, the DM shall not restart the debounce timer upon a further report of PREPASSED. ⌋*(SRS_Diag_04188)*

[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_04188)*

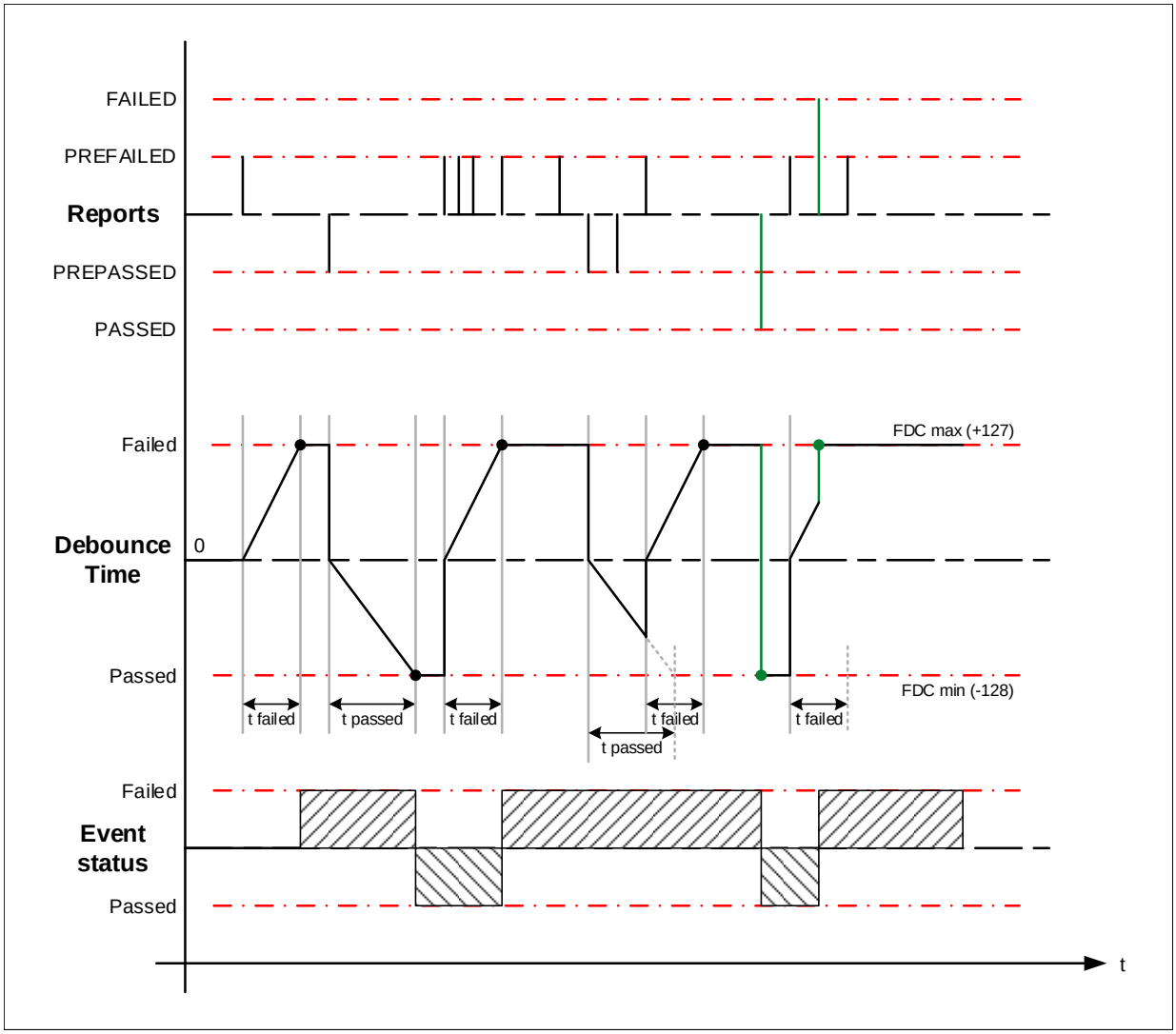

Figure 7.4: Timer based debouncing

[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_04188)*

7.2.1.4.4 Debounce algorithm reset

In some situations the application might want to reset the debouncing or to freeze it. The `DM` provides the interface `ResetEventDebounceStatus` 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_04188)*

[SWS_DM_00026] Application resetting the debounce counter ⌈ If the interface `ResetEventDebounceStatus` is called with `DebounceResetStatus` set to `RESET`, the `DM` shall reset the debounce counter. ⌋*(SRS_Diag_04188)*

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

[SWS_DM_00037] ⌈ If `ResetEventDebounceStatus` is called with `FREEZE`, the `DM` shall freeze the related debounce timer for event with configured timer based debouncing. ⌋*(SRS_Diag_04188)*

[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_04188)*

7.2.1.4.5 Dependencies to enable conditions

As described in chapter 7.2.4.3 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 ⌈ 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_04188)*

7.2.1.4.6 Dependencies to UDS service 0x85 ControlDTCSettings

[SWS_DM_00088] ControlDTCSetting influence ⌈ If `DiagnosticDebounceAlgorithmProps.debounceBehavior` set to `freeze`, the `DM` shall freeze the internal debounce timer when ControlDTCSetting is set to disabled for the related event. ⌋ *(SRS_Diag_04188)*

7.2.2 DTC Status processing

7.2.2.1 Status processing

[SWS_DM_00213] DTC status processing ⌈

The DM shall process the UDS status byte harmonizing with the ISO 14229-1 standard. ⌋*(SRS_Diag_04010)*

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 status byte triggered by the test results reported via the MonitorAction of the corresponding DiagnosticEvent service interface. ⌋ *(SRS_Diag_04010)*

Note that if configured, PREPASSED or PREFAILED status reports reported via MonitorAction trigger debounce mechanisms (see 7.2.1.4). These status reports do not have direct impact on the UDS status byte. If the status of an event gets fully qualified after debouncing (i.e. PASSED or FAILED), 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 status byte by setting **only** the 'testFailed' bit to 0 if if the event MonitorAction is set to RESET_TESTFAILED. ⌋*(SRS_Diag_04010)*

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 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 status byte triggered by the clearing of a DTC using the 0x14 ClearDiagnosticInformation UDS service. ⌋ *(SRS_Diag_04010)*

[SWS_DM_00218] Confirmation ⌈ The DM shall confirm the status of the UDS status byte by setting the 'confirmedDTC' bit to 1 if the threshold determined by the corresponding DiagnosticEvent.eventFailureCycleCounterThreshold configuration parameter is reached. ⌋*(SRS_Diag_04010)*

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 7.2.2.3.

7.2.2.2 Status change notifications

[SWS_DM_00219] Observability of the status byte ⌈ The DM shall provide the current status of events and DTCs for the AA via the CurrentEventStatus field of the corresponding DiagnosticEventNotification service interface and via the CurrentDTCStatus field of the corresponding DTCInformation service interface. ⌋*(SRS_Diag_04183)*

[SWS_DM_00220] Notification about the changes of the status byte ⌈

The DM shall inform the AA about the status changes of events and DTCs via the CurrentEventStatus field and EventStatusChanged event of the corresponding DiagnosticEventNotification service interface and via the CurrentDTCStatus field and DTCStatusChanged event of the corresponding DTCInformation service interface. ⌋*(SRS_Diag_04183)*

7.2.2.3 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 IndicatorStatus 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 corrsponding 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_04010)*

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 (Diagnostic-

`ConnectedIndicator.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)*

7.2.3 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 operation cycle State as started during the ECU startup and DM is initialisation. ⌋*(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 stop the respective operation cycles State 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 OperationCycleState set to END and take no further actions. ⌋*(SRS_Diag_04178)*

7.2.4 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_04150)*

[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_04150)*

7.2.4.1 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_04010)*

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

7.2.4.1.1 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. ⌋*(SRS_Diag_04010)*

[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

⌋()

7.2.4.1.2 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] Defintion of DTC groups ⌈ The existence of a DiagnosticTroubleCodeGroup shall define the existence of the DTC group with the DTC identifier DiagnosticTroubleCodeGroup.groupNumber ⌋(*SRS_Diag_04010*)

[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_04010*)

[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_04010*)

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.

7.2.4.2 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. ⌋*()*

7.2.4.3 EnableConditions

[SWS_DM_00072] Availability of enable condition service interfaces ⌈ The DM shall provide a service interface EnableCondition per configured DiagnosticEnableCondition. ⌋*(SRS_Diag_04192)*

DiagnosticEnableConditions are mapped to DiagnosticEvents by DiagnosticEventToEnableConditionGroupMappings.

[SWS_DM_00074] Unsatisfied enable conditions ⌈ If any of the enable conditions mapped to the event is not fulfilled, 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)*

7.2.4.4 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_00078] Unsatisfied storage conditions ⌈ If any of the storage conditions mapped to the event is not fulfilled, the DM shall ignore event data storage and updates. ⌋*(SRS_Diag_04192)*

[SWS_DM_00079] Fulfilled storage conditions ⌈ If all of the storage conditions mapped to the event are fulfilled, the DM shall allow the storage or respectively update of event related data. ⌋*(SRS_Diag_04192)*

7.2.4.5 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 exisits in the configuration)
- extended data if configured (at least one corresponding DiagnosticTroubleCodeProps.extendedDataRecord reference exisits in the configuration)

⌋*(SRS_Diag_04211)*

7.2.4.5.1 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.

7.2.4.5.2 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)*

[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)*

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 de-

fined by the DiagnosticTroubleCodeProps.freezeFrameContent configuration class. Each referenced DiagnosticDataIdentifier shall be captured in its order via the DataIdentifier service interface. ⌋(*SRS_Diag_04205*)

[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_04184*)

7.2.4.5.3 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*)

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 DiagnosticExtendedDataRecord.recordNumber configuration parameter defines the record number for each extended data record. ⌋(*SRS_Diag_04206*)

[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*)

7.2.4.6 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_04180*)

[SWS_DM_00117] Clearing a DTC ⌈ When the DM is about to clear a DTC it shall reset the event and UDS status bytes and clear the snapshot records and extended data records stored for this DTC. ⌋*(SRS_Diag_04180)*

7.2.4.6.1 Locking of the DTC clearing process by an client

The DM supports more than one diagnostic clients as described in chapter 7.1.3. 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_04180)*

[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_04180)*

[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_04180)*

[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_04180)*

7.2.4.6.2 Application permission to clear a DTC

In certain situations it is desirable to avoid that a DTC is cleared from the fault memory and an application can decide if a certain DTC can be cleared or not.

[SWS_DM_00118] Event specific configuration to allow clearing of a DTC ⌈ For all events having the DiagnosticEvent.eventClearAllowed set to requiresCallbackExecution the DM shall provide internal information if an event can be cleared or not. ⌋*(SRS_Diag_04191)*

[SWS_DM_00119] Init value for events with clear allowed information ⌈ Upon startup, the DM shall set all events having an event specific information to get cleared according to [SWS_DM_00118] to forbid the clearance. ⌋*(SRS_Diag_04191)*

Please note that the DM has a different semantics in interpretation of DiagnosticEvent.eventClearAllowed. While Autosar Diagnostic Extract Template [2]

mentions a callback from the diagnostic management to the application, the DM provides an interface for the application and it is up to the application to call the DM to allow the clearance of such an event. As by default the clear operation is forbidden for an event, the applications need to ensure that they allow the clearance of an event before a diagnostic clear DTC command is executed.

[SWS_DM_00120] Description of application interface to control the clear event behavior ⌈ If the interface SetClearAllowed is called the DM shall use the value of the parameter IsClearAllowed as the new clear allowed state. A value of "false" will forbid, a value of "true" will allow a clear DTC operation. ⌋*(SRS_Diag_04191)*

[SWS_DM_00125] Linking between event clear allowed and clearing a DTC ⌈ If one DiagnosticEventToTroubleCodeUdsMapping exists with DiagnosticEventToTroubleCodeUdsMapping.diagnosticEvent referencing an event with configured allowance to clear a DTC, the DM shall allow the clear the DTC referenced by DiagnosticEventToTroubleCodeUdsMapping.troubleCodeUds by evaluating the state of clear allow information. ⌋*(SRS_Diag_04191)*

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 and an DiagnosticEventToTroubleCodeUdsMapping exists with a mapping from this DTC to an event with a forbidden clear according to [SWS_DM_00120] and the event has DiagnosticEvent.clearEventBehavior set to noStatusByteChange, the DM shall not change the event and DTC status byte. ⌋*(SRS_Diag_04191)*

[SWS_DM_00124] Limited status byte clearing during a clear DTC operation ⌈ If the DM is requested to clear a DTC and an DiagnosticEventToTroubleCodeUdsMapping exists with a mapping from this DTC to an event with a forbidden clear according to [SWS_DM_00120] and the event has DiagnosticEvent.clearEventBehavior set to onlyThisCycleAndReadiness, the DM shall set the event and DTC status bytes:

- Bit 1 TestFailedThisOperationCycle
- Bit 4 TestNotCompletedSinceLastClear
- Bit 5 TestFailedSinceLastClear
- Bit 6 TestNotCompletedThisOperationCycle

to '0'. ⌋*(SRS_Diag_04191)*

[SWS_DM_00121] Forbidden clearing of snapshot records and extended data records ⌈ If the DM is requested to clear a DTC and an DiagnosticEventToTroubleCodeUdsMapping exists with a mapping from this DTC to an event with a forbidden clear according to [SWS_DM_00120], the DM shall leave all snapshot records and extended data records for this DTC unchanged. ⌋*(SRS_Diag_04191)*

7.2.4.6.3 DTC clearing triggered by application

Besides the UDS request ClearDiagnosticInformation according to 7.1.6.5.1 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 chapter 7.1.6.5.1. This could be realized using a dedicated diagnostic routine service, whose application is in charge of the clearing process.

[SWS_DM_00260] Singleton interface ClearDTC ⌈ The DM shall support one single instance of the `ClearDTC` interface. ⌋*(SRS_Diag_04194)*

[SWS_DM_00262] Common semantic behavior for clear 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 ⌈ If the method `Clear` of the interface `ClearDTC` is called, the DM shall clear the `DTC` or `DTC group` provided in the parameter `DTC`. The clear DTC shall clear the fault memory provided by the parameter `DTCOrigin`. ⌋*(SRS_Diag_04194)*

[SWS_DM_00263] ClearDTC call on invalid DTC or DTCgroup ⌈ If the method `Clear` of the interface `ClearDTC` 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_00264] ClearDTC call on invalid DTCOrigin ⌈ If the method `Clear` of the interface `ClearDTC` is called and the parameter `DTCOrigin` has no matching configured `DiagnosticMemoryDestination` exists, the DM shall trigger the error WRONG_DTCORIGIN. ⌋*(SRS_Diag_04194)*

[SWS_DM_00265] ClearDTC called while another clear operation is in progress ⌈ If the method `Clear` of the interface `ClearDTC` is called and another clear DTC operation is currently in progress for the selected `DTCOrigin`, 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 interface `ClearDTC` is called and the DM detects physical memory errors and thus cannot guarantee that the clear operation was done succesfully, the DM shall trigger the error MEMORY_ERROR. ⌋*(SRS_Diag_04194)*

[SWS_DM_00267] Possible failure of ClearDTC ⌈ If the method `Clear` of the interface `ClearDTC` 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)*

7.2.4.7 Aging

[SWS_DM_00237] Aging ⌈ The DM shall only support aging for events if the corresponding DiagnosticEvent.agingAllowed configuration parameter is set. ⌋ *(SRS_Diag_04010)*

[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_04010)*

[SWS_DM_00239] Aging counter ⌈ The DM shall support an aging counter for each event memory entry. ⌋*(SRS_Diag_04010)*

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. ⌋*(SRS_Diag_04010)*

[SWS_DM_00241] Aging cycle and threshold ⌈ If aging is supported for an event, the aging counter shall be calculated based on the DiagnosticAging.threshold and DiagnosticAging.agingCycle configuration parameters referenced by the DiagnosticTroubleCodeProps.aging configuration parameter of the DTC assigned to the event. The threshold defines the number of aging cycles in which the status was reported but not failed, this shall be calculated by the counter. After aging, the event memory entry shall be deleted (aged) from the event memory. ⌋ *(SRS_Diag_04010)*

[SWS_DM_00242] Reoccurrence after aging ⌈ The DM shall handle the reoccurrence of unlearned events like new events, since they were previously deleted from the event memory by aging. ⌋*(SRS_Diag_04010)*

[SWS_DM_00243] Aging-related UDS status byte processing ⌈ As a consequence of aging, the DM shall set 'testFailedSinceLastClear' and 'confirmedDTC' status bits to 0. ⌋*(SRS_Diag_04010)*

7.3 Required Configuration

The Autosar Diagnostic Extract Template [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 allows most of the elements to be optional and added at a later point in time. However at the point in time, when the Autosar Diagnostic Extract Template [2] is used to configure the DM, a certain minimum content is required. In this chapter a loose list of Autosar Diagnostic Extract Template [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`. ⌋*()*

[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`. ⌋*()*

[SWS_DM_CONSTR_00207] Required VINDataIdentifier ⌈ If DoIP is used as transport protocol according to [SWS_DM_00005] exactly one `DiagnosticDataIdentifier` with `representsVin` set to true shall exist in the configuration. ⌋*()*

According to [SWS_DM_00005] the DoIP transport layer is always implemented and [SWS_DM_CONSTR_00207] always applies. For completeness it is to mention that implementations without DoIP the presence of VINDataIdentifier is recommended, but not mandatory.

8 API specification

8.1 Type definitions

This chapter lists all types provided by the DM.

8.1.1 Diagnostic service management

8.1.1.1 DiagnosticSessionType

Name	DiagnosticSessionType		
Kind	Type		
Derived from	Texttable, uint8		
Description	Represents valid values for Diagnostic Sessions		
Range	DEFAULT_SESSION	0x01	UDS standardized default session
	PROGRAMMING_ SESSION	0x02	UDS standardized programming session
	EXTENDED_ DIAGNOSTIC_SESSION	0x03	UDS standardized extended diagnostic session
	SAFETY_SYSTEM_ DIAGNOSTIC_SESSION	0x04	UDS standardized safety system diagnostic session
	configuration dependent	0x40 - 0x7E	Symbol and Value can be deduced from DEXT Symbol = short-name of DiagnosticSession Value = DiagnosticSession.id

8.1.1.2 SecurityLevelType

Name	Security-Level Type
Kind	Type
Derived from	Texttable, uint8
Description	Represents valid values for Security-Levels

	SEC_LEV_LOCKED	0x00	No active unlocked security-level (default)
	SEC_LEV_ {DiagnosticSecurityLevel}	0x01 - 0x21	Security-Level assigned to Subfunctions requestSeed/sendKey (values: 0x01 - 0x42). Security-Level value corresponds to securityAccessType requestSeed value (0x01, 0x03, 0x05, .., 0x41) Variation: {DiagnosticSecurityLevel} = short name of DiagnosticSecurityLevel

8.1.1.3 MetaInfoKeyType

Name	MetaInfoKeyType
Kind	Type
Derived from	Texttable, uint8
Description	Represents the predefined/valid keys, which are available within the optional MetaInfo the DM provides in service processor calls.

	SA	0x00	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”
	TA	0x01	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”
	REQUEST_HANDLE	0x02	Key for the RequestHandle parameter of the current service request. The value in the MetaInf Map for this key, will be the stringified decimal representation of the RequestHandle.
	REQUEST_TYPE	0x03	Indicator whether request is functional or physical addressed. The value in the MetaInf Map for this key, will be either “PHYS” or “FUNC”.
	SUPP_POS_RESP	0x06	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”
	PROTOCOL_ID	0x07	Key for the ID of the Diagnostic Protocol to which the current request is assigned. The value in the MetaInf Map for this key, will be the stringified decimal representation of the protocol ID.

	DOIP_LOCAL_IP	0x08	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 multi-homed 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)
	DOIP_LOCAL_PORT	0x09	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.

	DOIP_REMOTE_IP	0x0A	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)
	DOIP_REMOTE_PORT	0x0B	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.

8.1.1.4 MetaInfoType

Name	MetaInfoType
Kind	Type
Derived from	Map<key=MetaInfoKeyType, value=variable-length-string>
Description	Meta-Inf map, which contains key-value pairs according to MetaInfoKeyType (see 8.1.1.3)

8.1.1.5 UDSResponseCodeType

Name	UDSResponseCodeType
Kind	Type
Derived from	Texttable, uint8
Description	Represents UDS Error Codes, which can be returned by the external service processor.

Range	GeneralReject	0x10	According to ISO.
	ServiceNotSupported	0x11	According to ISO.
	SubfunctionNotSupported	0x12	According to ISO.
	IncorrectMessageLength OrInvalidFormat	0x13	According to ISO.
	BusyRepeatRequest	0x21	According to ISO.
	ConditionsNotCorrect	0x22	According to ISO.
	RequestSequenceError	0x24	According to ISO.
	NoResponseFromSubnet Component	0x25	According to ISO.
	FailurePreventsExecutionOf RequestedAction	0x26	According to ISO.
	RequestOutOfRange	0x31	According to ISO.
	SecurityAccessDenied	0x33	According to ISO.
	invalidKey	0x35	According to ISO.
	exceedNumberOfAttempts	0x36	According to ISO.
	requiredTimeDelayNotExpired	0x37	According to ISO.
	uploadDownloadNotAccepted	0x70	According to ISO.
	transferDataSuspended	0x71	According to ISO.
	GeneralProgramming Failure	0x72	According to ISO.
	wrongBlockSequenceCounter	0x73	According to ISO.
	requestCorrectlyReceived ResponsePending	0x78	According to ISO.
	SubFunctionNotSupported InActiveSession	0x7E	According to ISO.
	ServiceNotSupported InActiveSession	0x7F	According to ISO.
	NoProcessingNoResponse	0xFF	Deviating from ISO - no further service processing and no response (silently ignore request message).

8.1.1.6 ConfirmationStatusType

Name	ConfirmationStatusType
Kind	Type
Derived from	Texttable, uint8
Description	Type used in the method confirmed for the status of the service processing

Range	RES_POS_OK	0x00	Positive response has been sent out successfully
	RES_POS_NOT_OK	0x01	Positive response has not been sent out successfully
	RES_NEG_OK	0x02	Negative response has been sent out successfully
	RES_NEG_NOT_OK	0x03	Negative response has not been sent out successfully
	RES_POS_SUPPRESSED	0x04	Positive answer suppressed

8.1.1.7 DiagnosticProtocolType

Name	DiagnosticProtocolType		
Kind	Type		
Derived from	Texttable, uint8		
Description	Represents the type of a Diagnostic Protocol.		
Range	UDS_ON_IP	0x05	Diagnostic Protocol according to ISO 14229-1[1], ISO14229-5 [9] and ISO 13400-2 [10]
	SUPPLIER_PROTOCOL_1	0xF0	Supplier-specific/proprietary protocol 1
	...	...	...
	SUPPLIER_PROTOCOL_15	0xFE	Supplier-specific/proprietary protocol 15

8.1.1.8 DiagnosticProtocolStatusType

Name	DiagnosticProtocolStatusType
Kind	Struct
Description	Represents the status of an Active Protocol

	Name	Type	Description
	ActivityStatus	Activity TypeSta-tus Type	ActivityStatus of this diagnostic protocol
	Session	Diagnostic Session Type	Diagnostic Session, which this Active Diagnostic Protocol is currently in.
	SecurityLevel	Security Level Type	Security-Level in which this Active Diagnostic Protocol is currently.

8.1.1.9 ActivityTypeStatusType

Name	ActivityTypeStatusType		
Kind	Type		
Derived from	Texttable, uint8		
Description	Type for ActivityType status		
Range	PROTOCOL_INACTIVE	0x00	Protocol is currently inactive
	PROTOCOL_ACTIVE	0x01	Diagnostic Protocol is currently activ i.e. activly processing a diagnostic request or is in any non-default session

8.1.1.10 DiagnosticProtocolIdentifierType

Name	DiagnosticProtocolIdentifierType		
Kind	Struct		
Description	Represents the status of an Active Protocol		
	Name	Type	Description
	DiagnosticProtocolType	Diagnostic Protocol Type	Diagnostic Protocol Type of the Active Protocol.
	TargetAddress	uint16	TargetAddress TA of tester currently active on this diagnostic protocol
	SourceAddress	uint16	SourceAddress SA of tester currently active on this diagnostic protocol

8.1.1.11 DiagnosticProtocolExtendedStatusType

Name	DiagnosticProtocolExtendedStatusType		
Kind	Struct		
Description	Represents the extended status of an Active Diagnostic Protocol		
	Name	***Type***	***Description***
	ControlDtcStatus	ControlDtcStatusType	ControlDtcStatus, stauts id DTC reporting is switched of via ControlDtcSetting service.
	CommunicationControlStatus	CommunicationControlStatusType	CommunicationControlStatus.

8.1.1.12 ControlDtcStatusType

Name	ControlDtcStatusType		
Kind	Type		
Derived from	Texttable, uint8		
Description	Type for ControlDTCStatus status		
Range	DTC_SETTING_ON	0x00	Updating of diagnostic trouble code status bits is under normal operating conditions
	DTC_SETTING_OFF	0x01	Updating of diagnostic trouble code status bits is stopped

8.1.2 Event memory management

8.1.2.1 MonitorActionType

Name	MonitorActionType
Kind	Type
Derived from	uint8
Description	Represents the status information reported by AAs being relevant for error monitoring.

Range	PASSED	0x00	Monitor reports qualified test result passed.
	FAILED	0x01	Monitor reports qualified test result failed.
	PREPASSED	0x02	Monitor reports qualified test result pre-passed.
	PREFAILED	0x03	Monitor reports qualified test result pre-failed.
	FDC_THRESHOLD_REACHED	0x04	Monitor triggers the storage of ExtendedDataRecords and FreezeFrames (if the triggering condition is connected to this threshold).
	RESET_TESTFAILED	0x05	Reset TestFailed Bit without any other side effects like readiness.
	FREEZE_DEBOUNCING	0x06	TBD.
	RESET_DEBOUNCING	0x07	TBD.
	PRESTORE	0x08	TBD.
	CLEAR_PRESTORE	0x09	TBD.
		0x10-0xFF	reserved

8.1.2.2 InitMonitorReasonType

Name	InitMonitorReasonType		
Kind	Type		
Derived from	uint8		
Description	Represents the information describing the reason of monitor reinitialization.		
Range	CLEAR	0x01	Event was cleared and all internal values and states are reset.
	RESTART	0x02	Operation cycle of the event was (re-)started.
	REENABLED	0x03	Enable conditions or DTC settings re-enabled.
	STORAGEREENABLED	0x04	Storage condition reenabled.
		0x00 0x05-0xFF	reserved

8.1.2.3 OperationCycleStateType

Name	OperationCycleStateType		
Kind	Type		
Derived from	uint8		
Description	Represents the state information of operation cycles.		
Range	START	0x00	Start/restart the operation cycle.
	END	0x01	End the operation cycle.
		0x02-0xFF	reserved

8.1.2.4 IndicatorStatusType

Name	IndicatorStatusType		
Kind	Type		
Derived from	uint8		
Description	Represents different modes of abstract indicators handled DM-internally.		
Range	OFF	0x00	Indicator off mode
	CONTINUOUS	0x01	Indicator continuously on mode
	BLINKING	0x02	Indicator blinking mode
	BLINK_CONT	0x03	Indicator blinking or continuously on mode
	SLOW_FLASH	0x04	Indicator slow flashing mode
	FAST_FLASH	0x05	Indicator fast flashing mode
	ON_DEMAND	0x06	Indicator on-demand mode
	SHORT	0x07	Indicator short mode

8.1.2.5 ClearFailedReasonType

Name	ClearFailedReasonType
Kind	Type
Derived from	uint8
Description	Represents the reason why processing a clear request failed.

Range	FAILED	0x03	Failed to clear the DTC due to any other reason
	BUSY	0x05	DTC not cleared, as another clearing process is in progress. The caller can retry later
	MEMORY_ERROR	0x06	An error occurred during erasing a memory location
	WRONG_DTC	0x08	DTC value does not exist in the current configuration
	WRONG_DTCORIGIN	0x09	Wrong DTC origin

8.1.2.6 EventStatusByteType

Name	EventStatusByteType			
Kind	Bitfield			
Derived from	uint8			
Lower limit	0x00			
Lower limit	0xFF			
Description	Represents the event status byte value. In this data-type each bit has an individual meaning. The bit is set to 1 when the condition holds. For example, if the 2nd bit (0x02) is set to 1, this means that the test failed this operation cycle. If the bit is set to 0, it has not yet failed this cycle.			
	Kind	***Name***	***Mask***	***Description***
	bit	TF	0x01	bit0: TestFailed
	bit	TFTOC	0x02	bit1: TestFailedThisOperationCycle
	bit	TNCTOC	0x40	bit6: TestNotCompletedThisOperationCycle

8.1.2.7 UdsStatusByteType

Name	UdsStatusByteType
Kind	Bitfield
Derived from	uint8
Lower limit	0x00
Lower limit	0xFF
Description	Represents the UDS status byte value. In this data-type each bit has an individual meaning. The bit is set to 1 when the condition holds. For example, if the 2nd bit (0x02) is set to 1, this means that the test failed this operation cycle. If the bit is set to 0, it has not yet failed this cycle.

Elements	Kind	Name	Mask	Description
	bit	TF	0x01	bit0: TestFailed
	bit	TFTOC	0x02	bit1: TestFailedThisOperationCycle
	bit	PDTC	0x04	bit2: PendingDTC
	bit	CDTC	0x08	bit3: ConfirmedDTC
	bit	TNCSLC	0x10	bit4: TestNotCompletedSinceLastClear
	bit	TFSLC	0x20	bit5: TestFailedSinceLastClear
	bit	TNCTOC	0x40	bit6: TestNotCompletedThisOperationCycle
	bit	WIR	0x80	bit7: WarningIndicatorRequested

8.1.2.8 SnapshotDataInfoType

Name	SnapshotDataInfoType		
Kind	Struct		
Description	Represents the contents of a snapshot data element with a snapshot record.		
	Name	Type	Description
	DataIdentifier	uint16	Data identifier assigned to this snapshot data info element. This value of this DID identifies the data and the content of the snapshot data stored.
	Data	Variable size array of uint8	Contains the snapshot data of the DID.

8.1.2.9 SnapshotRecordUpdatedType

Name	SnapshotRecordUpdatedType
Kind	Struct
Description	Represents the contents of a snapshot record.

	Name	Type	Description
	SnapshotRecordNumber	uint8	Identifies the snapshot record number assigned to the snapshot record.
	SnapshotDataElements	Variable size array of Snap-shot-DataIn-foType	Contains the snapshot data identifiers and the snapshot data of the snapshot record.

8.1.2.10 DTCStatusChangedType

Name	DTCStatusChangedType		
Kind	Struct		
Description	DTC update with old and new status.		
	Name	**Type**	**Description**
	DTC	uint32	Identifies the DTC number
	DTCStatusOld	UdsStatus ByteType	Contains the previous DTC status before the status update.
	DTCStatusNew	UdsStatus ByteType	Contains the current DTC status.

8.2 Service Interfaces

This chapter lists all provided and required service interfaces of the DM.

8.2.1 Diagnostic service management

8.2.1.1 DiagnosticProtocol

Port

Name	DiagnosticProtocol_{SoftwareCluster}_{DiagnosticProtocol}		
Kind	ProvidedPort	Interface	DiagnosticProtocol
Description	Provides information about diagnostic protocols.		
Variation	For each SoftwareCluster get all DiagnosticProtocols.		

Table 8.1: Port - DiagnosticProtocol_{SoftwareCluster}_{DiagnosticProtocol}

Service Interface

Name	DiagnosticProtocol

Table 8.2: Service Interfaces - DiagnosticProtocol

Method

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

Table 8.3: Service Interface DiagnosticProtocol - Method: Cancel

Fields

Name	ExtendedStatus
Description	Contains the current extended status (i.e. ControlDtcStatus) of the diagnostic protocol.
Type	DiagnosticProtocolExtendedStatusType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 8.4: Service Interface DiagnosticProtocol - Field: ExtendedStatus

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

Table 8.5: Service Interface DiagnosticProtocol - Field: Identifier

Name	Status
Description	Contains the current status (i.e. ActivityStatus, Session, SecurityLevel) of the diagnostic protocol.
Type	DiagnosticProtocolStatusType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 8.6: Service Interface DiagnosticProtocol - Field: Status

8.2.1.2 GenericUDSService

Port

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

Table 8.7: Port - GenericUDSService_{SoftwareCluster}_{DiagnosticServiceSwMapping}

Service Interface

Name	GenericUDSService	
Possible Errors	1	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.8: Service Interfaces - GenericUDSService

Method

Name	Service	
Description	Called for any request messsage of the mapped DiagnosticServiceSwMapping.	
Parameter	SID	
	Description	Diagnostic Request Service Identifier.
	Type	uint8
	Variation	-
	Direction	IN
Parameter	requestData	
	Description	Diagnostic request data.
	Type	DataArrayType
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Parameter	responseData	
	Description	Diagnostic response data.
	Type	DataArrayType
	Variation	-
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.9: Service Interface GenericUDSService - Method: Service

8.2.1.3 ServiceValidation

Port

Name	ServiceValidation_{SoftwareCluster}_{DiagnosticServiceSwMapping}		
Kind	RequiredPort	***Interface***	ServiceValidation
Description	This service is the manufacturer/supplier notification handler.		
Variation	For each SoftwareCluster.		

Table 8.10: Port - ServiceValidation_{SoftwareCluster}_{DiagnosticServiceSwMapping}

Service Interface

Name	ServiceValidation	
Possible Errors	1	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.11: Service Interfaces - ServiceValidation

Method

Name	Validate	
Description	Called for any request message.	
Parameter	requestData	
	Description	Diagnostic request message including SID.
	Type	DataArrayType
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.12: Service Interface ServiceValidation - 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.	
Parameter	status	
	Description	status/outcome of the service processing.
	Type	ConfirmationStatusType
	Variation	-
	Direction	IN

Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN

Table 8.13: Service Interface ServiceValidation - Method: Confirmation

8.2.1.4 DataIdentifier

Port

Name	DataIdentifier_{SoftwareCluster}_{DiagnosticDataIdentifier}		
Kind	RequiredPort	***Interface***	DataIdentifier_{Software Cluster}_{DiagnosticDataIdentifier}
Description	This is the default service interface for a DiagnosticDataIdentifier.		
Variation	For each SoftwareCluster get all DataIdentifiers.		

Table 8.14: Port - DataIdentifier_{SoftwareCluster}_{DiagnosticDataIdentifier}

Service Interface

Name	DataIdentifier_{SoftwareCluster}_{DiagnosticDataIdentifier}	
Variation	For each DiagnosticDataIdentifier get all DiagnosticDataIdentifier within each SoftwareCluster.	
Possible Errors	1	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.15: Service Interfaces - DataIdentifier

Method

Name	Read	
Description	Called for ReadDataByIdentifer request for this DiagnosticDataIdentifier (if configured).	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	DataElement within response message.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each DiagnsoticDataIdentifier get all its DiagnosticParameters. The Short-Name of this AttributeDataPrototype is Req_{DiagnosticData Element}.shortname.
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.16: 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: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	DataElement within the request message.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each DiagnsoticDataIdentifier get all its DiagnosticParameters. The Short-Name of this AttributeDataPrototype is Req_{DiagnosticData Element}.shortname.
	Direction	IN
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.17: Service Interface DataIdentifier - Method: Write

Name	Cancel	
Description	Method to cancle the current diagnostic protocol. This includes current request execution and reset of any protocol-specific states i.e. Session or Security.	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN

Table 8.18: Service Interface DataIdentifier - Method: Cancel

8.2.1.5 RoutineService

Port

Name	RoutineService_{SoftwareCluster}_{DiagnosticRoutine}		
Kind	RequiredPort	***Interface***	RoutineService_{Software Cluster}_{DiagnosticRoutine}
Description	Requires application providing methods of configured signature.		
Variation	For each SoftwareCluster get all DiagnosticRoutines.		

Table 8.19: Port - RoutineService_{SoftwareCluster}_{DiagnosticRoutine}

Service Interface

Name	RoutineService_{SoftwareCluster}_{DiagnosticRoutine}	
Variation	For each Routine get all DiagnosticRoutine within each SoftwareCluster.	
Possible Errors	1	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.20: Service Interfaces - RoutineService

Methods

Name	Start	
Description	Called for sub-function start of a routine.	
Variation	Let Method Routine.getAttribut("start").	
Parameter	req_{DiagnosticDataElement}	
	Description	IN-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each Method get all DiagnosticDataElements below attribute "request". The Short-Name of this AttributeDataPrototype is Req_{DiagnosticData Element}.shortname.
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each Method get all DiagnosticDataElements below attribute "response". The Short-Name of this AttributeDataPrototype is Resp_{DiagnosticData Element}.shortname.
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.21: Service Interface RoutineService - Method: Start

Name	Stop	
Description	Called for sub-function stop of a routine if configured.	
Variation	Let Method Routine.getAttribut("stop").	
Parameter	req_{DiagnosticDataElement}	
	Description	IN-Parameter of the stop sub-function according to DiagnosticRoutine.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each Method get all DiagnosticDataElements below attribute "request". The Short-Name of this AttributeDataPrototype is Req_{DiagnosticData Element}.shortname.
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each Method get all DiagnosticDataElements below attribute "response". The Short-Name of this AttributeDataPrototype is Resp_{DiagnosticData Element}.shortname.
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.22: Service Interface RoutineService - Method: Stop

Name	RequestResults	
Description	Called for sub-function stop of a routine if configured.	
Variation	Let Method Routine.getAttribut("requestResults").	
Parameter	req_{DiagnosticDataElement}	
	Description	IN-Parameter of the stop sub-function according to DiagnosticRoutine.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each Method get all DiagnosticDataElements below attribute "request". The Short-Name of this AttributeDataPrototype is Req_{DiagnosticData Element}.shortname.
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN

Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	For each Method get all DiagnosticDataElements below attribute "response". The Short-Name of this AttributeDataPrototype is Resp_{DiagnosticData Element}.shortname.
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.23: Service Interface RoutineService - Method: RequestResults

8.2.1.6 SecurityAccess

Port

Name	SecurityAccess_{SoftwareCluster}_{DiagnosticSecurityLevel}		
Kind	RequiredPort	***Interface***	ServiceValidation
Description	SecurityAccess.		
Variation	For each DiagnosticSecurityLevel within each SoftwareCluster .		

Table 8.24: Port - SecurityAccess_{SoftwareCluster}_{DiagnosticSecurityLevel}

Service Interface

Name	SecurityAccess	
Possible Errors	1	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.25: Service Interfaces - SecurityAccess

Methods

Name	GetSeed	
Description	Called for SecurityAccess (x027) with subfunction requestSeedId if configured (see Diagnostic SecurityAccess)	
Parameter	securityAccessDataRecord	
	Description	provided securityAccessDataRecord
	Type	DataArrayType
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Parameter	seed	
	Description	provided seed
	Type	DataArrayType
	Variation	-
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.26: Service Interface SecurityAccess - Method: GetSeed

Name	CompareKey
Description	Called for SecurityAccess (x027) with subfunction sendKey if configured (see DiagnosticSecurity Access).

Parameter	key	
	Description	The key to be validated
	Type	DataArrayType
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	MetaInfoType: Meta-Inf map, which contains key-value pairs according to Meta InfoKeyType.
	Variation	-
	Direction	IN
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.27: Service Interface SecurityAccess - Method: CompareKey

8.2.2 Event memory management

8.2.2.1 DiagnosticMonitor

Port

Name	DiagnosticMonitor_{SoftwareCluster}_{DiagnosticEvent}		
Kind	RequiredPort	***Interface***	DiagnosticMonitor
Description	Requires diagnostic monitor reporting results of diagnostic event monitoring.		
Variation	For each SoftwareCluster get all DiagnosticEvents of associated DEXT.		

Table 8.28: Port - DiagnosticMonitor_{SoftwareCluster}_{DiagnosticEvent}

Service Interface

Name	DiagnosticMonitor

Table 8.29: Service Interfaces - DiagnosticMonitor

Method

Name	InitMonitor	
Description	Event-specific notification for monitors about clearing, operation cycle restart or enable/storage condition re-enabling.	
Parameter	reason	
	Description	The reason for initiallizing the monitor.
	Type	InitMonitorReasonType
	Variation	-
	Direction	IN

Table 8.30: Service Interface DiagnosticMonitor - Method: InitMonitor

Event

Name	MonitorAction
Description	Reports monitor status information of an event.
Type	MonitorActionType

Table 8.31: Service Interface DiagnosticMonitor - Event: MonitorAction

8.2.2.2 DiagnosticEvent

Port

Name	DiagnosticEvent_{SoftwareCluster}_{DiagnosticEvent}		
Kind	ProvidedPort	***Interface***	DiagnosticEvent
Description	Provides information on diagnostic event.		
Variation	For each SoftwareCluster get all DiagnosticEvents of associated DEXT.		

Table 8.32: Port - DiagnosticEvent_{SoftwareCluster}_{DiagnosticEvent}

Service Interface

Name	DiagnosticEvent

Table 8.33: Service Interfaces - DiagnosticEvent

Field

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

Table 8.34: Service Interface DiagnosticEvent - Field: EventStatus

Methods

Name	FaultDetectionCounter	
Description	Returns the current value of fault detection counter of this event.	
Parameter	faultDetectionCounter	
	Description	Current FaultDetectionCounter value
	Type	sint8
	Variation	-
	Direction	OUT

Table 8.35: Service Interface DiagnosticEvent - Method: FaultDetectionCounter

Name	GetDebouncingOfEvent	
Description	Gets the debouncing status of an event.	
Parameter	debouncingState	
	Description	Current debouncing state
	Type	DebouncingStateType (similar to Dem_DebouncingStateType [SWS_Dem_01000])
	Variation	-
	Direction	OUT

Table 8.36: Service Interface DiagnosticEvent - Method: GetDebouncingOfEvent

Name	GetDTCOfEvent	
Description	Returns the DTC related to the event.	
Parameter	DTCFormat	
	Description	Define DTC format for the return value.
	Type	DTCFormatType (similar to Dem_DTCFormatType [SWS_Dem_00933])
	Variation	-
	Direction	IN
Parameter	DTCOfEvent	
	Description	DTC number in respective DTCFormatType
	Type	uint32
	Variation	-
	Direction	OUT

Table 8.37: Service Interface DiagnosticEvent - Method: GetDTCOfEvent

8.2.2.3 DTCInformation

Port

Name	DTCInformation_{SoftwareCluster}_{DiagnosticMemoryDestination}		
Kind	ProvidedPort	***Interface***	DTCInformation
Description	Provides information on diagnostic event.		
Variation	For each SoftwareCluster get all FaultMemories (DiagnosticMemoryDestinations) of associated DEXT.		

Table 8.38: Port - DTCInformation_{SoftwareCluster}_{DiagnosticMemoryDestination}

Service Interface

Name	DTCInformation

Table 8.39: Service Interfaces - DTCInformation

Method

Name	GetCurrentStatus	
Description	Determines the status of a DTC.	
Parameter	DTC	
	Description	DTC whose status is requested.
	Type	uint32
	Variation	-
	Direction	IN
Parameter	UDSStatusByte	
	Description	Contains the status of the DTC.
	Type	UdsStatusByteType
	Variation	-
	Direction	OUT

Table 8.40: Service Interface DTCInformation - Method: GetCurrentStatus

Events

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

Table 8.41: Service Interface DTCInformation - Event: DTCStatusChanged

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

Table 8.42: Service Interface DTCInformation - Event: SnapshotRecordUpdated

8.2.2.4 ClearDTC

Port

Name	ClearDTC_{SoftwareCluster}_{DiagnosticMemoryDestination}		
Kind	ProvidedPort	***Interface***	ClearDTC
Description	Provides method for clearing fault memory entries.		
Variation	For each SoftwareCluster get all FaultMemories (DiagnosticMemoryDestinations) of associated DEXT.		

Table 8.43: Port - ClearDTC_{SoftwareCluster}_{DiagnosticMemoryDestination}

Service Interface

Name	ClearDTC	
Possible Errors	1	errorContext of ClearFailedReason is of Type ClearFailedReasonType

Table 8.44: Service Interfaces - ClearDTC

Method

Name	Clear	
Description	Method for Clearing a DTC or a group of DTCs.	
Parameter	DTC	
	Description	DTC group to be cleared.
	Type	uint32
	Variation	-
	Direction	IN
Possible Application Errors	ClearFailed Reason	errorContext of ClearFailedReason is of Type ClearFailedReasonType

Table 8.45: Service Interface ClearDTC - Method: Clear

8.2.2.5 EnableCondition

Port

Name	EnableCondition_{SoftwareCluster}_{DiagnosticEnableCondition}		
Kind	ProvidedPort	***Interface***	EnableCondition
Description	Provides functionality for handling of enable conditions.		
Variation	For each SoftwareCluster get all DiagnosticEnableCondition of associated DEXT.		

Table 8.46: Port - EnableCondition_{SoftwareCluster}_{DiagnosticEnableCondition}

Service Interface

Name	EnableCondition

Table 8.47: Service Interfaces - EnableCondition

Field

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

Table 8.48: Service Interface EnableCondition - Field: State

8.2.2.6 StorageCondition

Port

Name	StorageCondition_{SoftwareCluster}_{DiagnosticStorageCondition}		
Kind	ProvidedPort	***Interface***	StorageCondition
Description	Provides functionality for handling of storage conditions.		
Variation	For each SoftwareCluster get all DiagnosticStorageCondition of associated DEXT.		

Table 8.49: Port - StorageCondition_{SoftwareCluster}_{DiagnosticStorageCondition}

Service Interface

Name	StorageCondition

Table 8.50: Service Interfaces - StorageCondition

Field

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

Table 8.51: Service Interface StorageCondition - Field: State

8.2.2.7 OperationCycle

Port

Name	OperationCycle_{SoftwareCluster}_{DiagnosticOperationCycle}		
Kind	ProvidedPort	***Interface***	OperationCycle
Description	Provides functionality for handling of operation cycles.		
Variation	For each SoftwareCluster get all DiagnosticOperationCycle of associated DEXT.		

Table 8.52: Port - OperationCycle_{SoftwareCluster}_{DiagnosticOperationCycle}

Service Interface

Name	OperationCycle

Table 8.53: Service Interfaces - OperationCycle

Field

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 8.54: Service Interface OperationCycle - Field: State

8.2.2.8 Indicator

Port

Name	Indicator_{SoftwareCluster}_{DiagnosticIndicator}		
Kind	ProvidedPort	***Interface***	Indicator
Description	Provides functionality for handling of indicators.		
Variation	For each SoftwareCluster get all DiagnosticIndicator of associated DEXT.		

Table 8.55: Port - Indicator_{SoftwareCluster}_{DiagnosticIndicator}

Service Interface

Name	Indicator

Table 8.56: Service Interfaces - Indicator

Field

Name	IndicatorStatus
Description	Contains the current state of an indicator.
Type	IndicatorStatusType
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 8.57: Service Interface Indicator - Field: IndicatorStatus

8.2.2.9 DataElement

Port

Name	DataElement_{SoftwareCluster}_{DiagnosticDataElement}		
Kind	RequiredPort	***Interface***	DataElement_{Software Cluster}_{DiagnosticDataElement}
Description	This is the service interface for any DiagnosticDataElement which is not used within a DataIdentifier service.		
Variation	For each SoftwareCluster get all DiagnosticDataElement in case a diagnostic software mapping exits		

Table 8.58: Port - DataElement_{SoftwareCluster}_{DiagnosticDataElement}

Service Interface

Name	DataElement_{SoftwareCluster}_{DiagnosticDataElement}
Variation	For each DiagnosticDataElement within each SoftwareCluster (only if service mapping exits).

Table 8.59: Service Interfaces - DataElement

Method

Name	Read	
Description	Called for data aquisition of a DiagnosticDataElement.	
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	OUT-Parameter.
	Type	DataType according to {DiagnosticDataElement}.swDataDefProps.
	Variation	-
	Direction	OUT

Table 8.60: Service Interface DataElement - Method: Read

8.3 C++ API Interfaces

This chapter lists all provided and required C++ API interfaces of the DM.

8.3.1 UDS Transportlayer C++ Interfaces

8.3.1.1 Provided C++ Interfaces

8.3.1.1.1 Common Types for the UDS Transportlayer C++ Interfaces

[SWS_DM_00335] Header file ⌈

The DM shall provide the definition of common types in `ara/diag/UdsTransportProtocolTypes.hpp` within namespace `ara::diag::udstransport`.⌋*()*

[SWS_DM_00336] UdsTransportProtocolHandlerID ⌈ The DM shall provide the following definition for UdsTransportProtocolHandlerID, which serves as the identifier of an Uds Transport Protocol implementation.

```
using UdsTransportProtocolHandlerID = uint8_t;
```

⌋()

[SWS_DM_00337] ChannelID ⌈ The DM shall provide the following definition for ChannelID, which serves as the identifier of a logical (network) channel, over which UDS messages can be sent/received.

```
using ChannelID = uint32_t;
```

⌋()

[SWS_DM_00338] ByteVector ⌈ The DM shall provide the following definition for ByteVector, which serves as the type of UDS message payloads.

```
using ByteVector = ... // e.g. std::vector<uint8_t>;
```

⌋()

[SWS_DM_00339] ByteVector vendor type ⌈ The type, the DM uses for ByteVector must behave like

```
std::vector<uint8_t>
```

⌋()

8.3.1.1.2 Class UdsMessage

The class UdsMessage represents an UDS message (request or response), which is exchanged between generic DM and an Uds Transport Protocol implementation.

[SWS_DM_00291] UdsMessage class ⌈ The DM shall provide an UdsMessage class in namespace ara::diag::udstransport.

```
class UdsMessage
```

⌋()

[SWS_DM_00292] UdsMessage non public constructors ⌈ The constructors of UdsMessage class must not be public. ⌋()

[SWS_DM_00293] UdsMessage Address type ⌈ UdsMessage shall provide a public typedef for adress type.

```
using Address = uint16_t;
```

⌋()

[SWS_DM_00294] meta info map type ⌈ UdsMessage shall provide a public typedef for the map of meta info.

```
using MetaInfoMap = ... // f.i. std::map<std::string, std::string>;
```

⌋()

[SWS_DM_00295] meta info map vendor type ⌈ The type, the DM uses for `MetaInfoMap` must behave like

```
std::map<std::string, std::string>
```

with regard to element access, iterators and capacity. ⌋()

[SWS_DM_00296] TargetAddressType Address type ⌈ `UdsMessage` shall provide a public enum class `TargetAddressType`

```
enum class TargetAddressType : std::uint8_t {
  kPhysical = 0, kFunctional = 1
};
```

⌋()

[SWS_DM_00297] GetSa method ⌈ `UdsMessage` shall provide a public method to access the source address of the UDS message.

```
Address GetSa() const;
```

⌋()

[SWS_DM_00298] GetTa method ⌈ `UdsMessage` shall provide a public method to access the target address of the UDS message.

```
Address GetTa() const;
```

⌋()

[SWS_DM_00299] GetTaType method ⌈ `UdsMessage` shall provide a public method to access the target address type of the UDS message.

```
TargetAddressType GetTaType() const;
```

⌋()

[SWS_DM_00300] GetPayload method readonly ⌈ `UdsMessage` shall provide a public method to access the payload of the UDS message starting with the `SID`.

```
const udstransport::ByteVector& GetPayload() const;
```

⌋()

[SWS_DM_00301] GetPayload method ⌈ `UdsMessage` shall provide a public method to access the payload of the UDS message starting with the `SID`.

```
udstransport::ByteVector& GetPayload();
```

⌋()

[SWS_DM_00302] AddMetaInfo method ⌈ `UdsMessage` shall provide a public method to add additional meta info to the existing meta info of the UDS message. The

meta info elements given in the parameter `meta_info` shall be added to UdsMessages internal meta info.

```
void AddMetaInfo(std::shared_ptr<const MetaInfoMap> meta_info);
```

⌋()

8.3.1.1.3 UdsMessage pointer definitions

[SWS_DM_00303] UdsMessage Pointer ⌈ DM shall provide a pointer to UDS message type in namespace `ara::diag::udstransport` of the following form:

```
using UdsMessagePtr = ...  // e.g. std::unique_ptr<UdsMessage, std::function<v
```

⌋()

[SWS_DM_00328] UdsMessage Pointer vendor type ⌈ The type, the DM uses for `UdsMessagePtr` must behave like

```
std::unique_ptr<UdsMessage>
```

⌋()

[SWS_DM_00304] Const UdsMessage Pointer ⌈ DM shall provide a pointer to constant UDS message type in namespace `ara::diag::udstransport` of the following form:

```
using UdsMessageConstPtr = ... // e.g. std::unique_ptr<const UdsMessage, std::
```

⌋()

[SWS_DM_00305] Const UdsMessage Pointer vendor type ⌈ The type, the DM uses for `UdsMessageConstPtr` must behave like

```
std::unique_ptr<const UdsMessage>
```

⌋()

8.3.1.1.4 Class UdsTransportProtocolMgr

The class `UdsTransportProtocolMgr` represents the facade the generic DM provides towards an `Uds Transport Protocol` implementation for UDS message exchange between the transport layer implementation and the transport layer implementation independent part of the DM.

[SWS_DM_00306] UdsTransportProtocolMgr class ⌈ The DM shall provide an `UdsTransportProtocolMgr` class in namespace `ara::diag::udstransport`.

```
class UdsTransportProtocolMgr
```

⌋()

[SWS_DM_00334] UdsTransportProtocolMgr may be an abstract class ⌈ The DM shall implement `UdsTransportProtocolMgr`. It shall be up to the implementation, whether `UdsTransportProtocolMgr` is an abstract class, which gets subclassed by DM or whether `UdsTransportProtocolMgr` is a non-abstract class.

```
class UdsTransportProtocolMgr
```

⌋()

[SWS_DM_00307] TransmissionResult type ⌈ The `UdsTransportProtocolMgr` shall provide an public enumeration type for an UDS message transmission result.

```
enum class TransmissionResult : std::uint8_t {
  kTransmitOk = 0, kTransmitFailed = 1
};
```

⌋()

[SWS_DM_00308] Global Channel Identifier type ⌈ The `UdsTransportProtocolMgr` shall provide an public typedef for a global channel identifier.

```
using GlobalChannelIdentifier = std::tuple<UdsTransportProtocolHandlerID, Chan
```

⌋()

[SWS_DM_00309] IndicateMessage method ⌈ The `UdsTransportProtocolMgr` shall provide a public method, where a `Uds Transport Protocol` implementation can indicate the reception of an UDS message.

```
virtual UdsMessagePtr IndicateMessage(
    UdsMessage::Address source_addr, UdsMessage::Address target_addr,
    UdsMessage::TargetAddressType type,
    GlobalChannelIdentifier global_channel_id, std::size_t size,
    std::shared_ptr<const UdsMessage::MetaInfoMap> meta_info)
```

⌋()

[SWS_DM_00310] NotifyMessageFailure method ⌈ The `UdsTransportProtocolMgr` shall provide a public method, where a `Uds Transport Protocol` implementation shall notify the generic DM, in case the UDS message indicated with `IndicateMessage` (see [SWS_DM_00309]) could not be successfully received.

```
virtual void NotifyMessageFailure(UdsMessageConstPtr message)
```

⌋()

[SWS_DM_00311] HandleMessage method ⌈ The `UdsTransportProtocolMgr` shall provide a public method, where a `Uds Transport Protocol` implementation shall hand over the successfully received UDS message to the generic DM for processing.

```
virtual void HandleMessage(UdsMessagePtr message)
```

⌋()

[SWS_DM_00312] TransmitConfirmation method ⌈ The UdsTransportProtocolMgr shall provide a public method, where a Uds Transport Protocol implementation shall inform the generic DM about the final outcome of a transmit call (see [SWS_DM_00327]).

```
virtual void TransmitConfirmation(UdsMessageConstPtr message,
                                  TransmissionResult result)
```

⌋()

[SWS_DM_00313] ChannelReestablished method ⌈ The UdsTransportProtocolMgr shall provide a public method, where a Uds Transport Protocol implementation shall inform the generic DM about the re-establishment of a diagnostic connection since the last start of the Uds Transport Protocol implementation (see [SWS_DM_00322]), in case the Uds Transport Protocol implementation has been instructed before (see [SWS_DM_00326]).

```
virtual void ChannelReestablished(GlobalChannelIdentifier global_channel_id)
```

⌋()

[SWS_DM_00314] HandlerStopped method ⌈ The UdsTransportProtocolMgr shall provide a public method, where a Uds Transport Protocol implementation shall inform the generic DM, that it has stopped. This is a callback of the Stop request (see [SWS_DM_00323]) called at the Uds Transport Protocol implementation.

```
virtual void HandlerStopped(UdsTransportProtocolHandlerID handler_id)
```

⌋()

8.3.1.1.5 Class UdsTransportProtocolHandler

The abstract class UdsTransportProtocolHandler represents the interface an Uds Transport Protocol implementation has to provide. The generic DM expects each Uds Transport Protocol to fulfill this API.

[SWS_DM_00315] UdsTransportProtocolHandler class ⌈ The DM shall provide an abstract UdsTransportProtocolHandler class in namespace ara::diag::udstransport.

```
class UdsTransportProtocolHandler
```

⌋()

[SWS_DM_00316] Header file ⌈ The DM shall provide the definition in ara/diag/UdsTransportProtocolHandler.hpp. ⌋()

[SWS_DM_00320] UdsTransportProtocolHandler UdsTransportProtocolMgr member ⌈ The UdsTransportProtocolHandler class shall provide a protected member of type UdsTransportProtocolMgr.

```
UdsTransportProtocolMgr& transportprotocol_manager_;
```

⌋()

[SWS_DM_00324] UdsTransportProtocolHandler UdsTransportProtocolHandlerID member ⌈ The `UdsTransportProtocolHandler` class shall provide a private member of type UdsTransportProtocolHandlerID.

```
const UdsTransportProtocolHandlerID handler_id_;
```

⌋()

[SWS_DM_00317] UdsTransportProtocolHandler constructor ⌈ The `UdsTransportProtocolHandler` class shall provide a constructor with the following signature:

```
explicit UdsTransportProtocolHandler(const UdsTransportProtocolHandlerID handler
```

⌋()

[SWS_DM_00321] constructor member initialization ⌈ The `UdsTransportProtocolHandler` constructor shall initialize its member `transportprotocol_manager_` ([SWS_DM_00320]) from its argument `transport_protocol_mgr` and its member `handler_id_` ([SWS_DM_00320]) from its argument `handler_id`.
⌋()

[SWS_DM_00318] UdsTransportProtocolHandler destructor ⌈ The `UdsTransportProtocolHandler` class shall provide a destructor with the following signature:

```
virtual ~UdsTransportProtocolHandler();
```

⌋()

[SWS_DM_00319] Initialize method ⌈ The `UdsTransportProtocolHandler` class shall provide a pure virtual method Initialize, which is called by `DM` to intialize the `Uds Transport Protocol` implementation.

```
virtual void Initialize() = 0;
```

⌋()

[SWS_DM_00322] Start method ⌈ The `UdsTransportProtocolHandler` class shall provide a pure virtual method Start, which is called by `DM` to Start the `Uds Transport Protocol` implementation, after it has been initialized.

```
virtual void Start() = 0;
```

⌋()

[SWS_DM_00323] Stop method ⌈ The `UdsTransportProtocolHandler` class shall provide a pure virtual method Stop, which is called by `DM` to Stop the `Uds Transport Protocol` implementation, after it has been started.

```
virtual void Stop() = 0;
```

⌋()

[SWS_DM_00325] GetHandlerID method ⌈ The `UdsTransportProtocolHandler` class shall provide a virtual method GetHandlerID, which returns the member `handler_id_` ([SWS_DM_00324])

```
virtual UdsTransportProtocolHandlerID GetHandlerID() const
```

⌋()

[SWS_DM_00326] NotifyReestablishment method ⌈ The `UdsTransportProtocolHandler` class shall provide a pure virtual method NotifyReestablishment, which is called by `DM` to trigger `Uds Transport Protocol` implementation, to notify `DM` in case the channel with the given `channel_id` is re-established after next call to `Start` (see [SWS_DM_00322]) via call to `ChannelReestablished` (see [SWS_DM_00313]).

```
virtual bool NotifyReestablishment(ChannelID channel_id) = 0;
```

⌋()

[SWS_DM_00327] Transmit method ⌈ The `UdsTransportProtocolHandler` class shall provide a pure virtual method Transmit, which is called by `DM` to trigger `Uds Transport Protocol` implementation, to transmit an UDS message on the given channel.

```
virtual void Transmit(UdsMessageConstPtr message, ChannelID channel_id) = 0;
```

⌋()

9 Sequence diagrams

9.1 Sequence Diagramms of UDS Transport Layer Interaction

9.1.1 Lifecycle

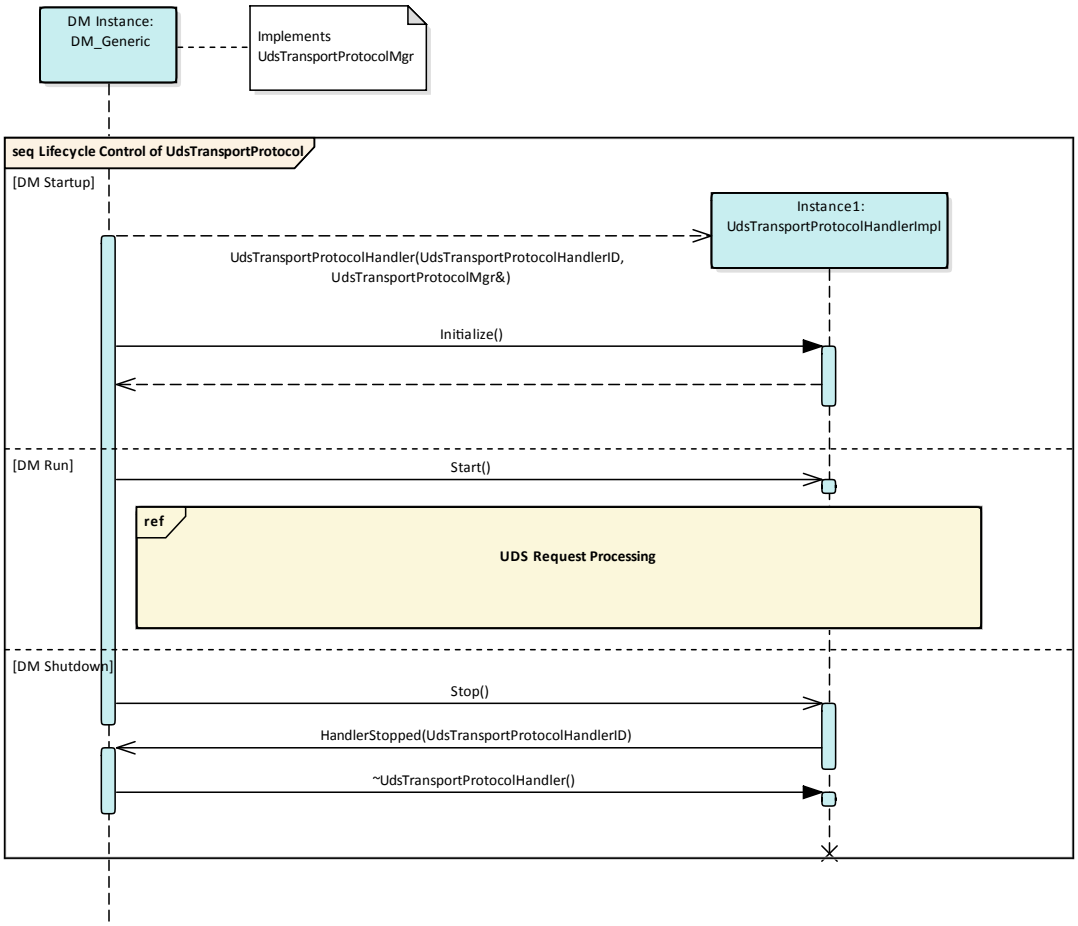

Figure 9.1: UDS Transport Lifecycle

9.1.2 UDS Request Processing

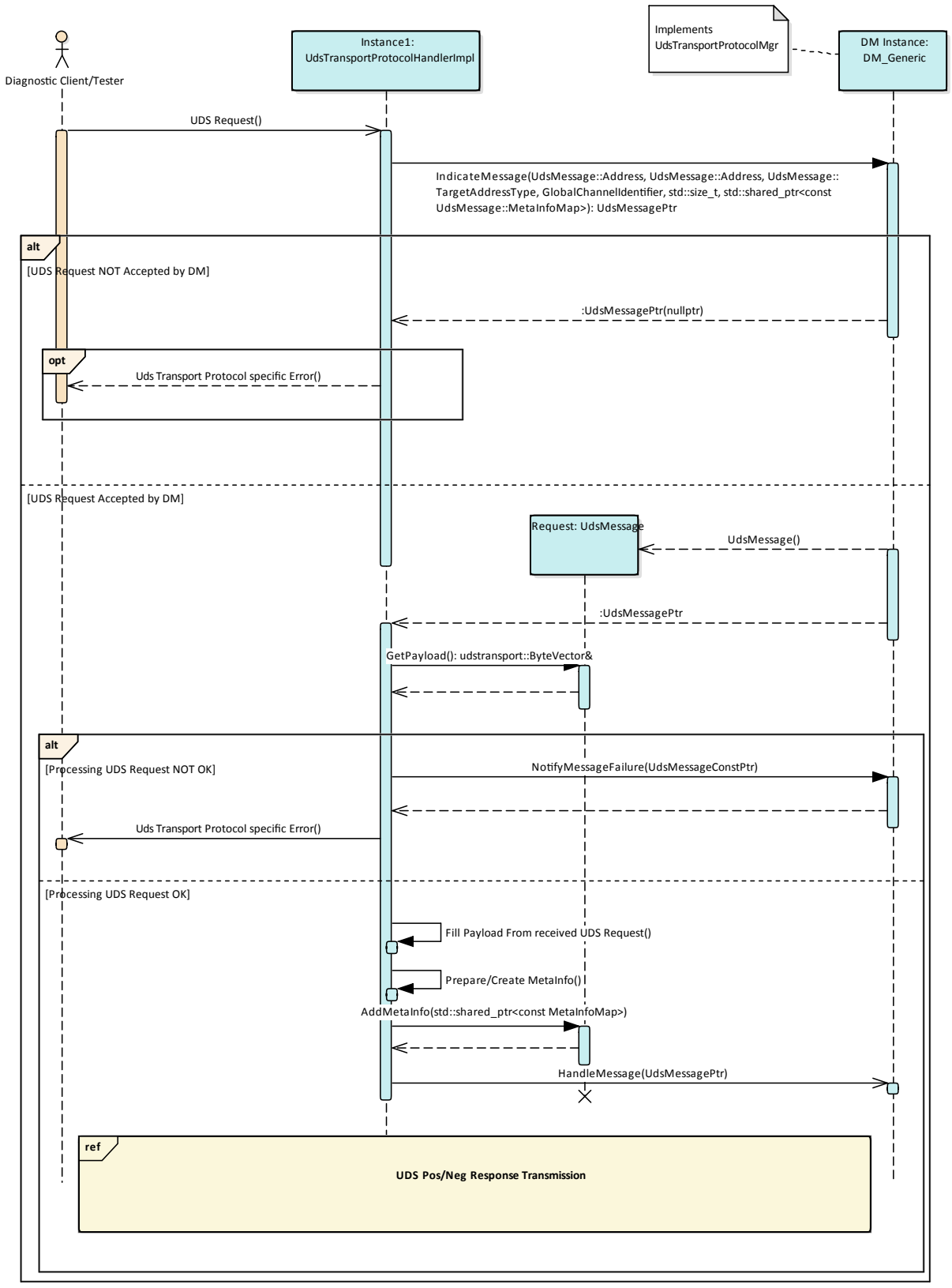

Figure 9.2: UDS Transport Request Processing

9.1.3 UDS Response Transmission

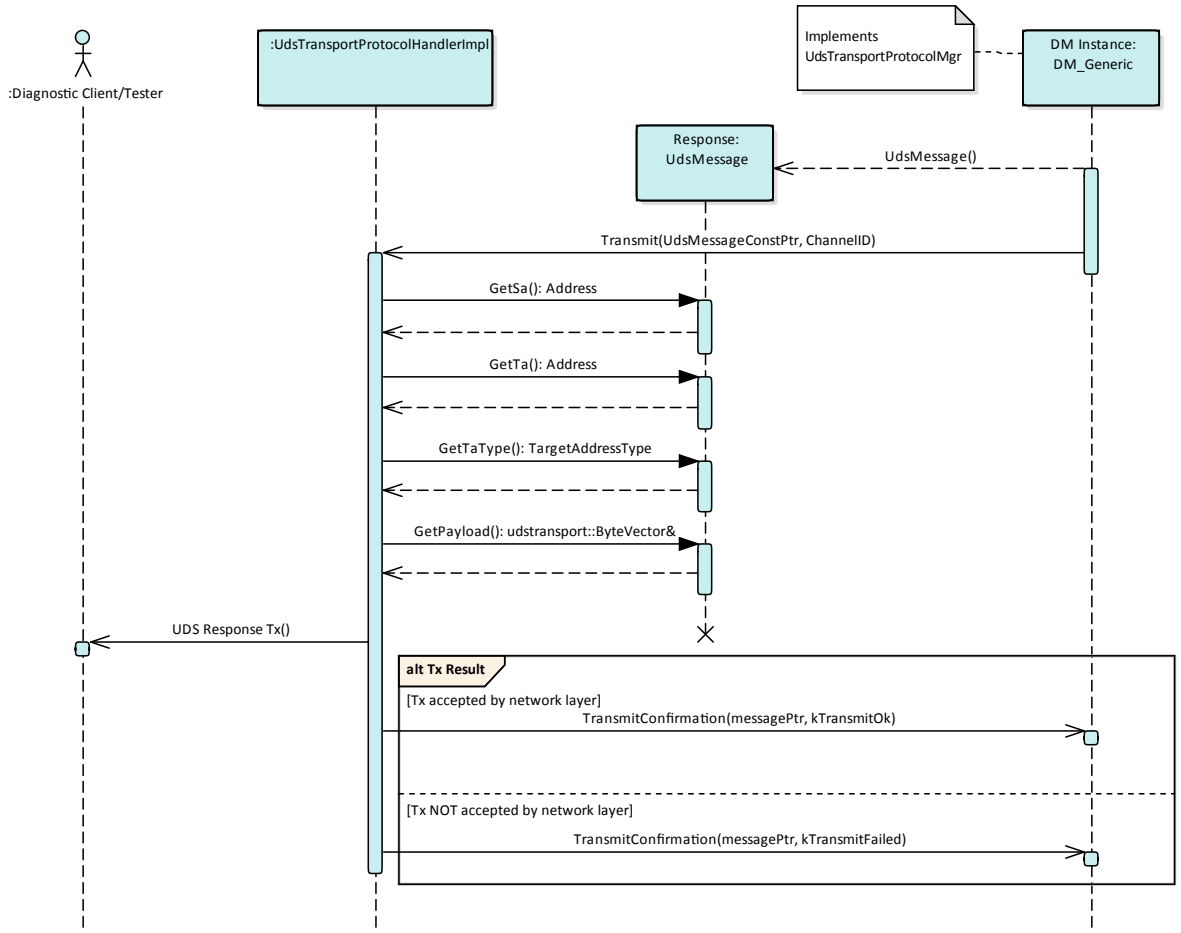

Figure 9.3: UDS Response Transmission

9.1.4 Channel Reestablishment

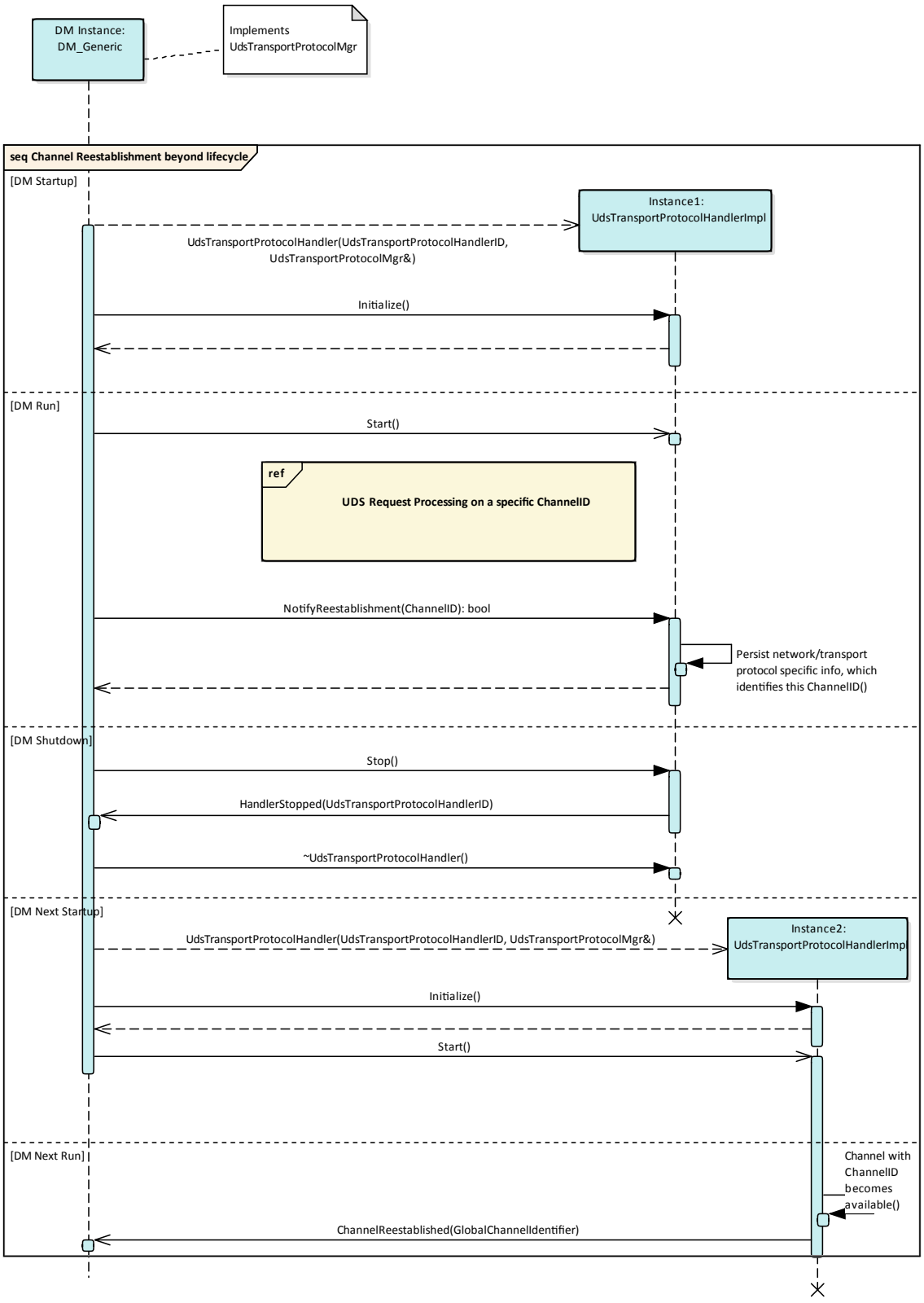

Figure 9.4: UDS Transport Channel Reestablishment

10 Configuration specification

In general, this chapter defines the configuration of the DM and the effects on the DM behaviour. The configuration is realized entirely using the Autosar Diagnostic Extract Template [2].

A Mentioned Class Tables

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 DemDebounceCounterBased.			
Base	ARObject, DiagEventDebounceAlgorithm, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
counterBasedFdcThresholdStorageValue	Integer	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.
counterDecrementStepSize	Integer	1	attr	This value shall be taken to decrement the internal debounce counter.
counterFailedThreshold	Integer	1	attr	This value defines the event-specific limit that indicates the "failed" counter status.
counterIncrementStepSize	Integer	1	attr	This value shall be taken to increment the internal debounce counter.
counterJumpDown	Boolean	1	attr	This value activates or deactivates the counter jump-down behavior.
counterJumpDownValue	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.
counterJumpUpValue	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from decrementing to incrementing.
counterPassedThreshold	Integer	1	attr	This value defines the event-specific limit that indicates the "passed" counter status.

Table A.1: 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 DemDebounceTimeBase.			
Base	ARObject, DiagEventDebounceAlgorithm, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
timeBasedFdcThresholdStorageValue	TimeValue	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.
timeFailedThreshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "failed" status.
timePassedThreshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "passed" status.

Table A.2: DiagEventDebounceTimeBased

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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticSession	DiagnosticSession	*	ref	This represents the associated DiagnosticSessions
environmentalCondition	DiagnosticEnvironmentalCondition	0..1	ref	This represents the environmental conditions associated with the access permission.
securityLevel	DiagnosticSecurityLevel	*	ref	This represents the associated DiagnosticSecurityLevels

Table A.3: DiagnosticAccessPermission

Class	**DiagnosticAging**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticAging			
Note	Defines the aging algorithm. **Tags:** atp.recommendedPackage=DiagnosticAgings			
Base	ARElement, ARObject, CollectableElement, DiagnosticCommonElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
agingCycle	DiagnosticOperationCycle	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.4: DiagnosticAging

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.5: DiagnosticClearDtcLimitationEnum

Enumeration	**DiagnosticClearEventBehaviorEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticEvent
Note	Possible behavior for clearing events.
Literal	***Description***
noStatusByteChange	The event status byte keeps unchanged. **Tags:** atp.EnumerationValue=0
onlyThisCycleAndReadiness	The OperationCycle and readiness bits of the event status byte are reset. **Tags:** atp.EnumerationValue=1

Table A.6: 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***
comControlClass	DiagnosticComControlClass	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.
customSubFunctionNumber	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.7: 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***
agingRequiresTestedCycle	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.
clearDtcLimitation	DiagnosticClearDtcLimitationEnum	1	attr	Defines the scope of the DEM_ClearDTC Api.
debounceAlgorithmProps	DiagnosticDebounceAlgorithmProps	*	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..*.

defaultEndianness	ByteOrderEnum	1	attr	Defines the default endianness of the data belonging to a DID or RID which is applicable if the DiagnosticDataElement does not define the endianness via the swDataDefProps.baseType attribute.
dtcStatusAvailabilityMask	PositiveInteger	1	attr	Mask for the supported DTC status bits by the Dem.
environmentDataCapture	DiagnosticDataCaptureEnum	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.
eventDisplacementStrategy	DiagnosticEventDisplacementStrategyEnum	1	attr	This attribute defines, whether support for event displacement is enabled or not, and which displacement strategy is followed.
maxNumberOfEventEntries	PositiveInteger	0..1	attr	This attribute fixes the maximum number of event entries in the fault memory.
maxNumberOfRequestCorrectlyReceivedResponsePending	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.
memoryEntryStorageTrigger	DiagnosticMemoryEntryStorageTriggerEnum	1	attr	Describes the primary trigger to allocate an event memory entry.
occurrenceCounterProcessing	DiagnosticOccurrenceCounterProcessingEnum	1	attr	This attribute defines the consideration of the fault confirmation process for the occurrence counter.
resetConfirmedBitOnOverflow	Boolean	1	attr	This attribute defines, whether the confirmed bit is reset or not while an event memory entry will be displaced.
responseOnAllRequestSids	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).
responseOnSecondDeclinedRequest	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 BusyRepeatRequest. FALSE: when the second request (Client B) can not be processed, it shall not be responded.

securityDelayTimeOnBoot	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.
statusBitHandlingTestFailedSinceLastClear	DiagnosticStatusBitHandlingTestFailedSinceLastClearEnum	1	attr	This attribute defines, whether the aging and displacement mechanism shall be applied to the "TestFailedSinceLastClear" status bits.
statusBitStorageTestFailed	Boolean	1	attr	This parameter is used to activate/deactivate the permanent storage of the "TestFailed" status bits. true: storage activated false: storage deactivated
typeOfDtcSupported	DiagnosticTypeOfDtcSupportedEnum	1	attr	This attribute defines the format returned by Dem_DcmGetTranslationType and does not relate to/influence the supported Dem functionality.
typeOfFreezeFrameRecordNumeration	DiagnosticTypeOfFreezeFrameRecordNumerationEnum	1	attr	This attribute defines the type of assigning freeze frame record numbers for event-specific freeze frame records.

Table A.8: 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	DiagnosticConnectedIndicatorBehaviorEnum	0..1	attr	Behavior of the linked indicator.
healingCycle	DiagnosticOperationCycle	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.9: DiagnosticConnectedIndicator

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	DiagnosticControlDTCSettingClass	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 DiagnosticControlDTCSetting in the given context.
dtcSettingParameter	PositiveInteger	1	attr	This represents the DTCSettingType defined by ISO 14229-1. The pre-defined values are 1 (ON) and 2 (OFF).

Table A.10: 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			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataIdentifier	DiagnosticAbstractDataIdentifier	1	ref	This represents the linked DiagnosticDataIdentifier.

Table A.11: 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***
arraySizeSemantics	ArraySizeSemanticsEnum	0..1	attr	This attribute controls the meaning of the value of the array size.
maxNumberOfElements	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.

swDataDefProps	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.12: 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, DiagnosticCommonElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataElement	DiagnosticParameter	1..*	aggr	This is the dataElement associated with the DiagnosticDataIdentifier. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataElement, variationPoint.shortLabel vh.latestBindingTime=postBuild
didSize	PositiveInteger	0..1	attr	This attribute indicates the size of the DiagnosticDataIdentifier.
representsVin	Boolean	0..1	attr	This attributes indicates whether the specific DiagnosticDataIdentifier represents the vehicle identification.
supportInfoByte	DiagnosticSupportInfoByte	0..1	aggr	This attribute represents the supported information associated with the DiagnosticDataIdentifier.

Table A.13: DiagnosticDataIdentifier

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.14: DiagnosticDebounceAlgorithmProps

Enumeration	**DiagnosticDebounceBehaviorEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticDebouncing Algorithm
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.15: DiagnosticDebounceBehaviorEnum

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***
customSubFunctionNumber	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	DiagnosticEcuResetClass	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.16: 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***
respondToReset	DiagnosticResponseToEcuResetEnum	0..1	attr	This attribute defines whether the response to the EcuReset service shall be transmitted before or after the actual reset.

Table A.17: 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.18: 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			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
compareType	DiagnosticCompareTypeEnum	1	attr	This attributes represents the concrete type of the comparison.

Table A.19: 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 DiagnosticEnvConditionFormula.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	DiagnosticLogicalOperatorEnum	1	attr	This attribute represents the concrete operator (supported operators: and, or) of the condition formula.
part (ordered)	DiagnosticEnvConditionFormulaPart	*	aggr	This aggregation represents the collection of formula parts that can be combined by logical operators.

Table A.20: 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.21: 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	DiagnosticEnvConditionFormula	1	aggr	This attribute represents the formula part of the DiagnosticEnvironmentalCondition.
modeElement	DiagnosticEnvModeElement	*	aggr	This aggregation contains a representation of ModeDeclarations in the context of a DiagnosticEnvironmentalCondition.

Table A.22: 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***
agingAllowed	Boolean	1	attr	This represents the decision whether aging is allowed for this DiagnosticEvent.
clearEventBehavior	DiagnosticClearEventBehaviorEnum	0..1	attr	This attribute defines the resulting UDS status byte for the related event, which shall not be cleared according to the ClearEventAllowed callback.
connectedIndicator	DiagnosticConnectedIndicator	*	aggr	Event specific description of Indicators. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
eventClearAllowed	DiagnosticEventClearAllowedEnum	0..1	attr	This attribute defines whether the Dem has access to a "ClearEventAllowed" callback.
eventFailureCycleCounterThreshold	PositiveInteger	0..1	attr	This attribute defines the number of failure cycles for the event based fault confirmation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild
eventKind	DiagnosticEventKindEnum	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 FreezeFrames is supported by the assigned event or not. True: Prestorage of FreezeFrames is supported False: Prestorage of FreezeFrames is not supported

Table A.23: DiagnosticEvent

Enumeration	**DiagnosticEventClearAllowedEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticEvent
Note	Denotes whether clearing of events is allowed.
Literal	***Description***
always	The clearing is allowed unconditionally. **Tags:** atp.EnumerationValue=0
requires Callback Execution	In case the clearing of a Diagnostic Event has to be allowed or prohibited through the SWC interface CallbackClearEventAllowed, the SWC has to indicate this by defining appropriate ServiceNeeds (i.e. DiagnosticEventNeeds). **Tags:** atp.EnumerationValue=2

Table A.24: DiagnosticEventClearAllowedEnum

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***
debounceAlgorithm	DiagnosticDebounceAlgorithmProps	1	ref	Reference to a DebounceAlgorithm assigned to a DiagnosticEvent.
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which a DebounceAlgorithm is assigned.

Table A.25: 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 EnableConditionGroup is assigned.
enableConditionGroup	DiagnosticEnableConditionGroup	1	ref	Reference to an EnableConditionGroup assigned to a DiagnosticEvent.

Table A.26: 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 OperationCycle is assigned.
operationCycle	DiagnosticOperationCycle	1	ref	Reference to an OperationCycle assigned to a DiagnosticEvent.

Table A.27: 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 StorageConditionGroup is assigned.
storageConditionGroup	DiagnosticStorageConditionGroup	1	ref	Reference to a StorageConditionGroup assigned to a DiagnosticEvent.

Table A.28: 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	DiagnosticTroubleCodeUds	1	ref	Reference to an UDS Diagnostic Trouble Code assigned to a DiagnosticEvent.

Table A.29: 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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
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	DiagnosticRecordTriggerEnum	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.30: 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***
recordNumber	PositiveInteger	0..1	attr	This attribute defines a record number for a freeze frame record. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

trigger	DiagnosticRecordTriggerEnum	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.31: 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
type	DiagnosticIndicatorTypeEnum	0..1	attr	Defines the type of the indicator.

Table A.32: DiagnosticIndicator

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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.33: 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, DiagnosticMemoryDestination, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.34: 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.35: 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.
cycleStatusStorage	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	DiagnosticOperationCycleTypeEnum	1	attr	Operation cycles types for the Dem to be supported by cycle-state APIs.

Table A.36: 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 DiagnosticParameter **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
supportInfo	DiagnosticParameterSupportInfo	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.37: DiagnosticParameter

Class	**DiagnosticProtocol**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticContribution			
Note	This meta-class represents the ability to define a diagnostic protocol. **Tags:** atp.recommendedPackage=DiagnosticProtocols			
Base	ARElement, ARObject, CollectableElement, DiagnosticCommonElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticConnection	DiagnosticConnection	*	ref	This represents the collection of applicable DiagnosticConnections for this DiagnosticProtocol. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=diagnosticConnection, variationPoint.shortLabel vh.latestBindingTime=postBuild
priority	PositiveInteger	1	attr	This represents the priority of the diagnostic protocol in comparison to other diagnostic protocols. Lower numeric values represent higher protocol priority: • 0 - Highest protocol priority • 255 - Lowest protocol priority
protocolKind	NameToken	1	attr	This identifies the applicable protocol.

serviceTable	DiagnosticServiceTable	0..1	ref	This represents the service table applicable for the given diagnostic protocol. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceTable, variationPoint.shortLabel vh.latestBindingTime=postBuild

Table A.38: DiagnosticProtocol

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***
readDTCInformationClass	DiagnosticReadDTCInformationClass	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 DiagnosticReadDTCInformation in the given context.

Table A.39: 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	DiagnosticReadDataByIdentifierClass	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 DiagnosticReadDataByIdentifier in the given context.

Table A.40: 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 DiagnosticReadDataByIdentifier.

Table A.41: 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
respondBeforeReset	Answer to EcuReset service should come before the reset. **Tags:** atp.EnumerationValue=1

Table A.42: 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	DiagnosticRequestRoutineResults	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	DiagnosticStart Routine	0..1	aggr	This represents the ability to start a routine
stop	DiagnosticStop Routine	0..1	aggr	This represents the ability to stop a running routine.

Table A.43: DiagnosticRoutine

Class	**DiagnosticSecurityAccess**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::Security Access			
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
securityAccessClass	DiagnosticSecurityAccessClass	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.44: DiagnosticSecurityAccess

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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
accessDataRecordSize	PositiveInteger	0..1	attr	This represents the size of the AccessDataRecord used in GetSeed. Unit:byte.
keySize	PositiveInteger	1	attr	This represents the size of the security key. Unit: byte.
numFailedSecurityAccess	PositiveInteger	1	attr	This represents the number of failed security accesses after which the delay time is activated.
securityDelayTime	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.45: 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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
accessPermission	DiagnosticAccessPermission	0..1	ref	This represents the collection of DiagnosticAccessPermissions that allow for the execution of the referencing DiagnosticServiceClass.
accessPermissionValidity	DiagnosticAccessPermissionValidityEnum	1	attr	This attribute is responsible for clarifying the validity of the accessPermission reference.

Table A.46: DiagnosticServiceClass

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. **Tags:** atp.recommendedPackage=DiagnosticServiceMappings			
Base	ARElement, ARObject, CollectableElement, DiagnosticCommonElement, DiagnosticMapping, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticDataElement	DiagnosticDataElement	1	ref	This represents the applicable payload that corresponds to the referenced DataPrototype in the role mappedDataElement.
mappedApDataElement	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
mappedDataElement	DataPrototype	0..1	iref	This represents the dataElement in the application software that is accessed for diagnostic purpose.

Table A.47: 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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
accessPermission	DiagnosticAccessPermission	0..1	ref	This represents the collection of DiagnosticAccessPermissions 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 DiagnosticServiceInstance. The subclasses that affected by this pattern implement references to the applicable "class"-role that substantiate this abstract reference. **Stereotypes:** atpAbstract

Table A.48: 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***
diagnosticDataElement	DiagnosticDataElement	0..1	ref	This represents a DiagnosticDataElement required to execute the respective diagnostic service in the context of the diagnostic service mapping,
mappedBswServiceDependency	BswServiceDependencyIdent	0..1	ref	This is supposed to represent a reference to a BswServiceDependency. 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.
mappedFlatSwcServiceDependency	SwcServiceDependency	0..1	ref	This represents the ability to refer to an AtomicSwComponentType that is available without the definition of how it will be emebdded into the component hiearchy.
mappedSwcServiceDependencyInExecutable	SwcServiceDependency	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
mappedSwcServiceDependencyInSystem	SwcServiceDependency	0..1	iref	This represents the ability to point into the component hiearchy (under possible consideration of the rootSoftwareComposition)
serviceInstance	DiagnosticServiceInstance	0..1	ref	This represents the service instance that needs to be considered in this diagnostics service mapping.

Table A.49: 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, MultilanguageReferrable, 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

jumpToBootLoader	DiagnosticJumpToBootLoaderEnum	1	attr	This attribute represents the ability to define whether this diagnostic session allows to jump to Bootloader (OEM Bootloader or System Supplier Bootloader). If this diagnostic session doesn't allow to jump to Bootloader the value JumpToBootLoaderEnum.noBoot shall be chosen.
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.
p2StarServerMax	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.50: DiagnosticSession

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			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.51: 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, MultilanguageReferrable, 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.52: 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, MultilanguageReferrable, 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.
environmentCaptureToReporting	EnvironmentCaptureToReportingEnum	0..1	attr	This attribute determines the point in time, when the data actually is captured.
extendedDataRecord	DiagnosticExtendedDataRecord	*	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
freezeFrameContent	DiagnosticDataIdentifierSet	0..1	ref	This represents the content of the a set of DiagnosticFreezeFrames.
immediateNvDataStorage	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

maxNumberFreezeFrameRecords	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.
memoryDestination	DiagnosticMemoryDestination	*	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	DiagnosticSignificanceEnum	0..1	attr	Significance of the event, which indicates additional information concerning fault classification and resolution.

Table A.53: 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***
considerPtoStatus	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	DiagnosticTroubleCodeProps	0..1	ref	Defined properties associated with the DemDTC.
eventObdReadinessGroup	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	DiagnosticUdsSeverityEnum	0..1	attr	DTC severity according to ISO 14229-1.
udsDtcValue	PositiveInteger	0..1	attr	Unique Diagnostic Trouble Code value for UDS.
wwhObdDtcClass	DiagnosticWwhObdDtcClassEnum	0..1	attr	This attribute is used to identify (if applicable) the corresponding severity class of an WWH-OBD DTC.

Table A.54: DiagnosticTroubleCodeUds

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	DiagnosticWriteDataByIdentifierClass	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 DiagnosticWriteDataByIdentifier in the given context.

Table A.55: 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			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
desc	MultiLanguageOverviewParagraph	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
introductio n	Documentation Block	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.56: Identifiable

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 DataTypes in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNativeTypeQualifier • 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***
additionalNativeTypeQualifier	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
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
implementationDataType	ImplementationDataType	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 SwPointerTargetProps), 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 memoryAllocationKeywordPolicy of the referenced SwAddrMethod. **Tags:** xml.sequenceOffset=33
swBitRepresentation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxisSet	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
swComparisonVariable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170; xml.typeElement=false
swDataDependency	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

swIntendedResolution	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
swInterpolationMethod	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
swPointerTargetProps	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecordLayout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290
swRefreshTiming	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
swValueBlockSize	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
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
valueAxisDataType	ApplicationPrimitiveDataType	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.57: SwDataDefProps