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	18-03

Document Change History

Date	Release	Changed by	Description
2018-03-29	18-03	AUTOSAR Release Management	• Chapter 7.1. Software Cluster added • Chapter 7.2. Diagnostic Service Management, common parts for all services separated • Chapter 7.3. Event Management, several additions and rework • Chapter 8. API specification, complete rework
2017-10-27	17-10	AUTOSAR Release Management	• General API rework • TP Plug-in interface • Introduction of SoftwareCluster in APIs • Additional UDS services like SecurityAccess
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

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

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

1.1 AUTOSAR Diagnostic Extract Template (DEXT)

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

1.2 Software Cluster

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

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

2 Acronyms and Abbreviations

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

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

Abbreviation / Acronym:	Description:
DID	Data Identified according to ISO 14229-1[1]. This 16 bit value uniquely defines one ore more data elements (parameters) that can are used in diagnostics to read, write or control data.
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.
PowerMode	Vehicle basic status information retrieval of DoIP
SA	Source Address of a UDS request
SID	Service Identifier, identifying a diagnostic service according to UDS, such as 0x14 ClearDiagnosticInformation
UDS	Unified Diagnostic Services
VIN	Vehicle Identification Number according to ISO-3779
Dcm	Diagnostic Communication Manager (Module of the AUTOSAR Classic Platform)
DoIP	Diagnostics over Internet Protocol (Communication protocol of automotive electronics according to ISO-13400[4])

Terms:	Description:
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.
Associated ServiceInterface	Describes the association of a `ServiceInterface` to a `DiagnosticServiceSwMapping` by means of a referenced `SwcServiceDependency`, see section 7.5.2.1.
DCM/DEM exclusive	Classifies an `internal DiagnosticDataElement` to be defined exclusively in the context of `Diagnostic Protocol`, called DCM-exclusive, or exclusively in the context of `Event Memory`, called DEM-exclusive.
Diagnostic Protocol	Diagnostic Protocol is an ISO-14229 term, describing the diagnostic conversation between a distinguishable diagnostic client and the diagnostic server.
Diagnostic Server	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).
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.

Terms:	Description:
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.
Internal, External	Classifies if a `DiagnosticDataElement` is either managed internally inside `DM` or by an external adaptive applications, see 7.5.1 for the precise definition.
Internally, Externally	Definition of the support type of a SID by the DM. Internally means processing is done by DM itself, Externally means an external service processor is used.
Monitor	A monitor is a piece of software running within an application, monitoring the correct functionality of a certain system part. The result of such a function check is reported to the DM in form of an diagnostic `event`.
Operation cycle	An operation cycle is the execution of monitor within an application, from a start point to a defined end point inside the application run.
Primary event memory	The primary event memory is used to store events and event related data. It is typically used by OEMs for after sales purposes, containing information to repair the vehicle.
Snapshot Record	Set of measurement values stored in the fault memory at a certain point of time during fault detection. It is used to gain environmental data information for occurred faults.
SoftwareCluster	A SoftwareCluster groups all AUTOSAR artifacts which are relevant to deploy software on a machine. This includes the definition of applications, i.e. their executables, application manifests, communication and diagnostics. In the context of diagnostics a SoftwareCluster can be addressed individually by its own set of diagnostic addresses.
UDS service	A diagnostic service as defined in ISO 14229-1[1].
UDS 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 Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[6] General Specification of Adaptive Platform
AUTOSAR_SWS_General

[7] Specification of Log and Trace for Adaptive Platform
AUTOSAR_SWS_AdaptiveLogAndTrace

[8] Requirements on Diagnostic
AUTOSAR_SRS_Diagnostic

[9] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

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

3.2 Related specification

AUTOSAR provides a specification of the general requirements that are specific for the adaptive platform [5, RS General], which is valid for for the DM.

AUTOSAR provides a General Specification on Basic Software modules [6, 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:
  - 0x23 ReadMemoryByAddress
  - 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 (reportNumberOfEmissionsOBDDTCByStatusMask)
  - 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.

[SWS_DM_00001] SRS Diagnostics ⌈ These items are currently not implemented. ⌋*(SRS_Diag_04059, SRS_Diag_04064, SRS_Diag_04171, SRS_Diag_04195, SRS_Diag_04200, SRS_Diag_04202, SRS_Diag_04218)*

5 Dependencies to other modules

DM is a service and therefore uses ara::com to communicate with applications. The DM uses ara::log ([7], 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 [8] and links to the fulfilling requirements by this document. Please note that the column “Satisfied by” being empty for a specific requirement means that the requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_Diag_04005]	Manage Security Access level handling	[SWS_DM_00047] [SWS_DM_00236] [SWS_DM_00411]
[SRS_Diag_04006]	Manage session handling	[SWS_DM_00046] [SWS_DM_00380] [SWS_DM_00381] [SWS_DM_00382] [SWS_DM_00383] [SWS_DM_00410]
[SRS_Diag_04016]	Support ”Busy handling” by sending a negative response 0x78	[SWS_DM_00368] [SWS_DM_00369]
[SRS_Diag_04019]	Provide confirmation after transmit diagnostic responses to the application	[SWS_DM_00268] [SWS_DM_00341]
[SRS_Diag_04020]	Suppress responses to diagnostic tool requests	[SWS_DM_00365] [SWS_DM_00366]
[SRS_Diag_04033]	Support the upload/download services for reading/writing data in an ECU in an extended and manufacturer specific diagnostic session	[SWS_DM_00128] [SWS_DM_00136] [SWS_DM_00138] [SWS_DM_00139] [SWS_DM_00142] [SWS_DM_00143]
[SRS_Diag_04059]	Configuration of timing parameters	[SWS_DM_00001]
[SRS_Diag_04064]	Provide configurable buffer sizes for storage of the events, status information and environmental data	[SWS_DM_00001]
[SRS_Diag_04067]	Provide the diagnostic status information according to ISO 14229-1	[SWS_DM_00061] [SWS_DM_00062] [SWS_DM_00063] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374]

Requirement	Description	Satisfied by
[SRS_Diag_04068]	Event specific debounce algorithms	[SWS_DM_00013] [SWS_DM_00014] [SWS_DM_00015] [SWS_DM_00026] [SWS_DM_00030] [SWS_DM_00031] [SWS_DM_00032] [SWS_DM_00033] [SWS_DM_00034] [SWS_DM_00035] [SWS_DM_00036] [SWS_DM_00037] [SWS_DM_00038] [SWS_DM_00039] [SWS_DM_00040] [SWS_DM_00085] [SWS_DM_00086] [SWS_DM_00089]
[SRS_Diag_04097]	Decentralized and modular diagnostic configuration in applications	[SWS_DM_00393] [SWS_DM_00397] [SWS_DM_00401] [SWS_DM_00402] [SWS_DM_00403] [SWS_DM_00404] [SWS_DM_00405] [SWS_DM_00406] [SWS_DM_00407] [SWS_DM_00408] [SWS_DM_00418] [SWS_DM_CONSTR_00394] [SWS_DM_CONSTR_00395] [SWS_DM_CONSTR_00396]
[SRS_Diag_04115]	The optional parameter DTCSettingControlOption Record as part of UDS service ControlDTCSetting shall be limited to GroupOfDTC	[SWS_DM_00231]
[SRS_Diag_04117]	Configurable behavior for DTC deletion	[SWS_DM_00065] [SWS_DM_00091] [SWS_DM_00092] [SWS_DM_00116] [SWS_DM_00117] [SWS_DM_00118] [SWS_DM_00119] [SWS_DM_00120] [SWS_DM_00121] [SWS_DM_00122] [SWS_DM_00123] [SWS_DM_00124] [SWS_DM_00125] [SWS_DM_00144] [SWS_DM_00145] [SWS_DM_00146] [SWS_DM_00147] [SWS_DM_00159] [SWS_DM_00160] [SWS_DM_CONSTR_00082]
[SRS_Diag_04119]	Handle the execution of diagnostic services according to the assigned diagnostic session	[SWS_DM_00046]
[SRS_Diag_04120]	Support a predefined Address AndLengthFormatIdentifier	[SWS_DM_00129] [SWS_DM_00130]
[SRS_Diag_04124]	Store the current debounce counter value non-volatile to over a power-down cycle	[SWS_DM_00018] [SWS_DM_00028]
[SRS_Diag_04125]	Event debounce counter shall be configurable	[SWS_DM_00017] [SWS_DM_00019] [SWS_DM_00020] [SWS_DM_00021] [SWS_DM_00022] [SWS_DM_00023] [SWS_DM_00024] [SWS_DM_00025] [SWS_DM_00029]
[SRS_Diag_04127]	Configurable record numbers and trigger options for DTCSnapshotRecords and DTCExtendedDataRecords	[SWS_DM_00153] [SWS_DM_00156]
[SRS_Diag_04133]	Aging for event memory entries	[SWS_DM_00237] [SWS_DM_00238] [SWS_DM_00239] [SWS_DM_00240] [SWS_DM_00241] [SWS_DM_00242]

Requirement	Description	Satisfied by
[SRS_Diag_04140]	Aging for UDS status bits ”confirmedDTC” and ”testFailed SinceLastClear”	[SWS_DM_00243]
[SRS_Diag_04148]	Provide capabilities to inform applications about diagnostic data changes	[SWS_DM_00273]
[SRS_Diag_04150]	Support the primary fault memory defined by ISO 14229-1	[SWS_DM_00056] [SWS_DM_00083] [SWS_DM_CONSTR_00084]
[SRS_Diag_04151]	Event status handling	[SWS_DM_00213] [SWS_DM_00214] [SWS_DM_00215]
[SRS_Diag_04157]	Reporting of DTCs and related data	[SWS_DM_00061] [SWS_DM_00062] [SWS_DM_00063] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00247] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374]
[SRS_Diag_04159]	Control of DTC storage	[SWS_DM_00088] [SWS_DM_00229] [SWS_DM_00232] [SWS_DM_00233] [SWS_DM_00378]
[SRS_Diag_04166]	Several tester conversations in parallel with assigned priorities	[SWS_DM_00011] [SWS_DM_00016] [SWS_DM_00051] [SWS_DM_00052] [SWS_DM_00180] [SWS_DM_00182] [SWS_DM_00183] [SWS_DM_00184] [SWS_DM_00185]
[SRS_Diag_04167]	Conversation preemption/abortion	[SWS_DM_00042] [SWS_DM_00049] [SWS_DM_00051] [SWS_DM_00052] [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_00329] [SWS_DM_00330] [SWS_DM_00331] [SWS_DM_00332] [SWS_DM_00333] [SWS_DM_00340] [SWS_DM_00342] [SWS_DM_00345] [SWS_DM_00346] [SWS_DM_00347] [SWS_DM_00348] [SWS_DM_00349] [SWS_DM_00350] [SWS_DM_00351] [SWS_DM_00356] [SWS_DM_00357] [SWS_DM_00358] [SWS_DM_00359] [SWS_DM_00385] [SWS_DM_00386] [SWS_DM_00387] [SWS_DM_00388] [SWS_DM_00389] [SWS_DM_00392]
[SRS_Diag_04169]	Provide an interface for external UDS service processors.	[SWS_DM_00197]
[SRS_Diag_04171]	Synchronous and asynchronous interaction with external service processors	[SWS_DM_00001]
[SRS_Diag_04172]	Inform external service processors about outcome of the final response	[SWS_DM_00341]

Requirement	Description	Satisfied by
[SRS_Diag_04178]	Support operation cycles according to ISO 14229-1	[SWS_DM_00002] [SWS_DM_00003] [SWS_DM_00004] [SWS_DM_00167] [SWS_DM_00169] [SWS_DM_00192] [SWS_DM_00216] [SWS_DM_CONSTR_00168]
[SRS_Diag_04179]	Provide interfaces for monitoring application.	[SWS_DM_00007] [SWS_DM_00008] [SWS_DM_00166] [SWS_DM_00168]
[SRS_Diag_04180]	Process all UDS Services related to diagnostic fault memory of ISO 14229-1 internally	[SWS_DM_00062] [SWS_DM_00090] [SWS_DM_00091] [SWS_DM_00092] [SWS_DM_00104] [SWS_DM_00115] [SWS_DM_00161] [SWS_DM_00162] [SWS_DM_00163] [SWS_DM_00164] [SWS_DM_00165] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00229] [SWS_DM_00232] [SWS_DM_00233] [SWS_DM_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00247] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374]
[SRS_Diag_04183]	Notify interested parties about event status changes	[SWS_DM_00219] [SWS_DM_00220]
[SRS_Diag_04185]	Notify applications about the clearing of an event	[SWS_DM_00066] [SWS_DM_00067]
[SRS_Diag_04186]	Notify applications about the start or restart of an operation cycle	[SWS_DM_00066] [SWS_DM_00068] [SWS_DM_00069] [SWS_DM_00070] [SWS_DM_00071]
[SRS_Diag_04189]	Support a fine grained configuration for Snapshot Records and ExtendedData Records	[SWS_DM_00151] [SWS_DM_00155]
[SRS_Diag_04190]	Usage of internal data elements in SnapshotRecords and ExtendedDataRecords	[SWS_DM_00152] [SWS_DM_00154]
[SRS_Diag_04192]	Provide the ability to handle event specific enable conditions	[SWS_DM_00074] [SWS_DM_00087] [SWS_DM_00377] [SWS_DM_00379]
[SRS_Diag_04194]	ClearDTC shall be accessible for applications	[SWS_DM_00260] [SWS_DM_00261] [SWS_DM_00262] [SWS_DM_00263] [SWS_DM_00265] [SWS_DM_00266] [SWS_DM_00267]
[SRS_Diag_04195]	Chronological reporting order of the DTCs located in the configured event memory	[SWS_DM_00001]
[SRS_Diag_04196]	UDS Service handling for all diagnostic services defined in ISO 14229-2	[SWS_DM_00090] [SWS_DM_00096] [SWS_DM_00097] [SWS_DM_00104] [SWS_DM_00113] [SWS_DM_00114] [SWS_DM_00126] [SWS_DM_00127] [SWS_DM_00128] [SWS_DM_00131] [SWS_DM_00134] [SWS_DM_00137] [SWS_DM_00140] [SWS_DM_00141] [SWS_DM_00161] [SWS_DM_00162] [SWS_DM_00170] [SWS_DM_00177] [SWS_DM_00186] [SWS_DM_00198] [SWS_DM_00199] [SWS_DM_00201] [SWS_DM_00210] [SWS_DM_00211]

Requirement	Description	Satisfied by
		[SWS_DM_00212] [SWS_DM_00227] [SWS_DM_00234] [SWS_DM_00235] [SWS_DM_00236] [SWS_DM_00269] [SWS_DM_00274] [SWS_DM_00360] [SWS_DM_00361] [SWS_DM_00363] [SWS_DM_00364] [SWS_DM_00367] [SWS_DM_00376] [SWS_DM_00419]
[SRS_Diag_04197]	Clearing the user defined fault memory	[SWS_DM_00193] [SWS_DM_00194] [SWS_DM_00195] [SWS_DM_00208]
[SRS_Diag_04198]	Process all UDS Services related to session and security management of ISO 14229 internally	[SWS_DM_00104] [SWS_DM_00226] [SWS_DM_00228]
[SRS_Diag_04199]	Provide a configurable UDS service execution mechanism at runtime to decide if a UDS request shall be processed or not	[SWS_DM_00105] [SWS_DM_00106] [SWS_DM_00107] [SWS_DM_00108] [SWS_DM_00111] [SWS_DM_00112] [SWS_DM_00286] [SWS_DM_00287] [SWS_DM_00288] [SWS_DM_00289]
[SRS_Diag_04200]	Support event combination	[SWS_DM_00001]
[SRS_Diag_04201]	Support a configuration to assign specific events to a customer specific DTC	[SWS_DM_00060] [SWS_DM_CONSTR_00059]
[SRS_Diag_04202]	Report DTCs getting active to the error logging module/system	[SWS_DM_00001]
[SRS_Diag_04203]	Common checks on all supported UDS Services Requests	[SWS_DM_00096] [SWS_DM_00098] [SWS_DM_00099] [SWS_DM_00100] [SWS_DM_00101] [SWS_DM_00102] [SWS_DM_00103] [SWS_DM_00202] [SWS_DM_00203] [SWS_DM_00230] [SWS_DM_00231] [SWS_DM_00249] [SWS_DM_00252] [SWS_DM_00362] [SWS_DM_00409] [SWS_DM_00412] [SWS_DM_00413] [SWS_DM_00414] [SWS_DM_00415] [SWS_DM_00416] [SWS_DM_00417]
[SRS_Diag_04204]	Provide the current status of each warning indicator.	[SWS_DM_00221] [SWS_DM_00222] [SWS_DM_00223] [SWS_DM_00224]
[SRS_Diag_04205]	Support of SnapshotRecords	[SWS_DM_00151] [SWS_DM_00152] [SWS_DM_00153]
[SRS_Diag_04206]	Support of ExtendedData Records	[SWS_DM_00154] [SWS_DM_00155] [SWS_DM_00156]
[SRS_Diag_04208]	Inform the application about diagnostic session and diagnostic security level changes on each tester connection.	[SWS_DM_00248] [SWS_DM_00250] [SWS_DM_00270] [SWS_DM_00271] [SWS_DM_00272]
[SRS_Diag_04209]	Pseudo parallel client interaction according to ISO	[SWS_DM_00011] [SWS_DM_00041] [SWS_DM_00043] [SWS_DM_00044] [SWS_DM_00045] [SWS_DM_00258] [SWS_DM_00259]
[SRS_Diag_04210]	Fully parallel client interaction	[SWS_DM_00011] [SWS_DM_00048]
[SRS_Diag_04211]	Persistent storage of DTC status and environmental data	[SWS_DM_00148] [SWS_DM_00150]

Requirement	Description	Satisfied by
[SRS_Diag_04214]	Support the user defined fault memories defined by ISO 14229-1	[SWS_DM_00055] [SWS_DM_00057]
[SRS_Diag_04216]	Support for multiple Diagnostic Server Instances	[SWS_DM_00390] [SWS_DM_00391] [SWS_DM_00420]
[SRS_Diag_04218]	Support of UDS service 0x2F InputOutputControlByIDentifier.	[SWS_DM_00001]
[SRS_Eth_00026]	No description	[SWS_DM_00205] [SWS_DM_00434] [SWS_DM_CONSTR_00206] [SWS_DM_CONSTR_00207]
[SRS_Eth_00081]	No description	[SWS_DM_00012]
[SRS_Eth_00083]	No description	[SWS_DM_00005]
[TPS_DEXT_01008]	No description	[SWS_DM_00058]
[TPS_DEXT_03014]	No description	[SWS_DM_00064]

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 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 SoftwareClusters. To support the current approaches of diagnostic management (like software updates), each SoftwareCluster has its own diagnosticAddresss. For details on the semantics and precise configuration of SoftwareClusters, see [9].

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) and each Diagnostic Server manages its own resources.

[SWS_DM_00420] Instantiation of Diagnostic Server ⌈ DM shall instantiate an independent Diagnostic Server per configured SoftwareCluster with dedicated resources. ⌋*(SRS_Diag_04216)*

7.2 Diagnostic service management

7.2.1 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.
- Dispatching the request towards the Diagnostic Server instances depending on target address and target address type (physical or functional) of received UDS request message
- Correlating the diagnostic request to an existing UDS session (if already exists)
- Checking whether the diagnostic request is allowed within current session and security settings
- If diagnostic request is NOT allowed, generate negative UDS response and send it to the network layer
- If diagnostic request is allowed, depending on DM's configuration and request type,
  - either process the service internally within diagnostic service handling function block of DM
  - or process the service internally within event memory management function block of DM
  - or hand it over for processing to an (external to DM) Adaptive Application

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

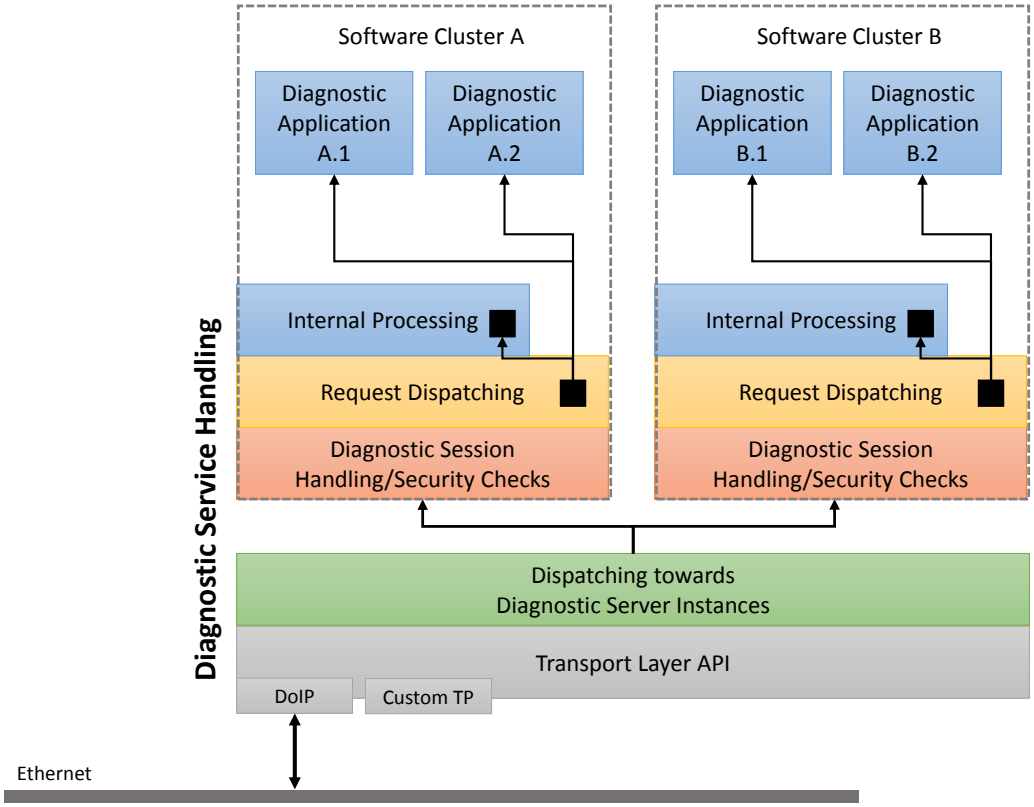

Figure 7.1: Architecture Diagnostic Service Handling

7.2.2 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.

7.2.2.1 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_Eth_00083)*

[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 DiagnosticDataIdentifier.representsVin set to true. ⌋(*SRS_Eth_00026*)

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].

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

7.2.2.2 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].

7.2.2.2.1 Initialization, Starting and Stopping of a proprietary UDS TransportLayer

[SWS_DM_00329] Lifecycle management of an Uds Transport Protocol implementation ⌈ The lifecycle of an Uds Transport Protocol implementation, 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])

⌋(*SRS_Diag_04168*)

[SWS_DM_00330] Construction of an Uds Transport Protocol implementation ⌈ The DM shall call the specific constructor of the Uds Transport Protocol implementation, where the argument handler_id is unique among all by DM instantiated Uds Transport Protocol implementations and the trans-

port_protocol_mgr is set to the reference of the instance of UdsTransportProtocolMgr (see [SWS_DM_00306]) provided by DM. ⌋(*SRS_Diag_04168*)

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

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

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

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

7.2.2.2.2 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

⌋(*SRS_Diag_04168*)

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

[SWS_DM_00385] Acceptance of UDS message reception ⌈ If the DM is able to process the indicated request, it shall return a std::pair with IndicationResult set to kIndicationOk and a UdsMessagePtr, which owns a valid UdsMessage

object, with a capacity of so many bytes, the DM wants to process of the indicated request. It shall be at least one byte. ⌋(SRS_Diag_04168)

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

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

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

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

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

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

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

[SWS_DM_00345] Forwarding of UDS message ⌈ If the Uds Transport Protocol implementation has filled the payload of the returned UdsMessagePtr, it shall

call HandleMessage ([SWS_DM_00311]) on its UdsTransportProtocolMgr reference ((see [SWS_DM_00330]) with the returned UdsMessagePtr as argument. ⌋ *(SRS_Diag_04168)*

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

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

7.2.2.2.3 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. ⌋*(SRS_Diag_04168)*

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

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

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

7.2.2.2.4 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. ⌋ *(SRS_Diag_04168)*

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

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

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

7.2.2.3 Dispatching of UDS Requests

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

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

7.2.3 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` (7.2.3.5) or `pseudo parallel client concept` (7.2.3.4). ⌋*(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)*

7.2.3.1 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 request (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_04119, SRS_Diag_04006)*

[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_04005)*

7.2.3.2 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[10]) 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_Eth_00081)*

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.

7.2.3.3 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)*

[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)*

[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)*

[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)*

[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)*

[SWS_DM_00280] Cancellation of Active Protocol at 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 the transmit functionality of the UDS TransportLayer was already called, nothing has to be done by DM, i.e. the response will be effectively sent out. ⌋*(SRS_Diag_04167)*

[SWS_DM_00281] Cancellation of active DiagnosticConversation in Non-Default Session ⌈ If DM decides to cancel a DiagnosticConversation with the activityStatus kActive according to [SWS_DM_00051], in case the Diagnostic-

Conversations diagnosticSession is set to non-default, the DM shall reset the DiagnosticConversations diagnosticSession to default and update the field Status of that DiagnosticConversation. ⌋(*SRS_Diag_04167*)

[SWS_DM_00282] Handling of non-/active diagnostic conversations ⌈ The DM shall set the activityStatus of a canceled DiagnosticConversation in the Status field to kInactive. ⌋(*SRS_Diag_04167*)

[SWS_DM_00042] Cancelling external service processors ⌈ External service processors, which are connected via the service interfaces mentioned in 8.2, shall be canceled by a call to the corresponding Cancel method if it is defined for the respective interface. ⌋(*SRS_Diag_04167*)

7.2.3.4 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_04209*)

[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_04209*)

[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_04209*)

[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_00281], and [SWS_DM_00282] and process the new request. ⌋*(SRS_Diag_04209)*

[SWS_DM_00259] Completion of already `Active Protocol`s 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_04209)*

Note: This means, that in this case also in 'pseudo parallel concept' there is a short timeframe until all `Active Protocol`s in default session are completed, where the `DM` at the same time has `Active Protocol`s 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_04209)*

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.

7.2.3.5 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 7.2.3.4. 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 Protocol`s) 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 Protocol`s is already reached and no cancellation ac-

cording to [SWS_DM_00051] applies, the DM shall refuse the request according to [SWS_DM_00049] and [SWS_DM_00290]. ⌋*(SRS_Diag_04210)*

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

7.2.3.6 Protocol Prioritization and Cancellation

Both 'Parallel Client Handling Variants' depicted in 7.2.3.4 and 7.2.3.5 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_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.

7.2.3.7 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_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,

- Protocol Kind according to `diagnosticProtocolKind`,
- Network Endpoint Source Address of the client (format depends on `diagnosticProtocolKind`).

⌋(*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_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_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

7.2.4 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]. ⌋(*SRS_Diag_04196, SRS_Diag_04203*)

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 7.2.3)
- **manufacturer specific failure detected?** Decision by applying manufacturer specific checks according to 7.2.4.4
- **SID supported?** Decision according to 7.2.4.2
- **SID supported in active session?** Decision according to 7.2.4.3

- **SID security check o.k.?** Decision according to 7.2.4.3
- **supplier-specific failure detected?** Decision by applying supplier-specific checks according to 7.2.4.4

[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. ⌋*(SRS_Diag_04196)*

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

7.2.4.1 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)*

7.2.4.2 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)*

7.2.4.3 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*)

7.2.4.4 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 8.2.1.4), 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 8.2.1.4), 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, SRS_Diag_04172)*

7.2.4.5 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 a condition check when the ISO 14229-1[1] mentions a service specific "Condition check" in the defined NRC handling for a given diagnostic service. The DM shall send the configured NRC value (see [SWS_DM_00289]) if the condition is not fulfilled. ⌋ *(SRS_Diag_04199)*

[SWS_DM_00112] Condition check definition ⌈ The 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_04199)*

[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 [2]). ⌋*(SRS_Diag_04199)*

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

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

[SWS_DM_00288] Configurable environmental condition check evaluates to TRUE ⌈ If the computation of the DiagnosticEnvConditionFormula evaluated to TRUE, the 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)*

7.2.5 Assemble positive or negative response

7.2.5.1 Positive Response

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

7.2.5.2 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)*

7.2.5.3 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 UDSResponseCodeType:

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

and the request is functional addressed, the ApplicationError shall be suppressed. ⌋*(SRS_Diag_04020)*

7.2.5.4 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)*

7.2.5.5 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 and send a negative response with NRC 0x10 (General reject). ⌋*(SRS_Diag_04016)*

7.2.6 Keep track of active non-default sessions

[SWS_DM_00380] Support for S3 timer ⌈ The `DM` shall provide support for $S3_{Server}$ (session timeout) with a fixed value of 5 second. The timer handling shall be implemented according to ISO 14229-1. ⌋*(SRS_Diag_04006)*

[SWS_DM_00381] Session timeout ⌈ Whenever a non-default session is active and when the session timeout ($S3_{Server}$) is reached without receiving any diagnostic request, the `DM` shall reset to the default session state. `DM` internal states for service processing shall be reset according to ISO 14229-1. ⌋*(SRS_Diag_04006)*

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

- Completion of any final response message or an error indication during sending of the response ([SWS_DM_00312]/`TransmitConfirmation`)
- Completion of the requested action in case no response message (positive and negative) is required / allowed.
- In case of an error during the reception of a multi-frame request message ([SWS_DM_00310]/ `NotifyMessageFailure`)

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

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

7.2.7 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 DiagnosticServiceClass referenced by a DiagnosticServiceInstance.serviceClass. ⌋*(SRS_Diag_04196)*

7.2.7.1 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.

7.2.7.2 Common service processing items

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

[SWS_DM_00410] Check session permission ⌈ A UDS Service request shall be evaluated against session execution permission using the DiagnosticAccessPermission referenced by the DiagnosticServiceInstance associated to the given request. A session execution permission check is passed if and only if one of the following holds:

- there are no DiagnosticSessions referenced in the role of diagnosticSession,
- there exists one DiagnosticSession referenced in the role of diagnosticSession with id matching the given session.

⌋(*SRS_Diag_04006*)

[SWS_DM_00411] Check security level permission ⌈ A UDS Service request shall be evaluated against security level execution permission using the DiagnosticAccessPermission referenced by the DiagnosticServiceInstance associated to the given request. A security level execution permission check is passed if and only if one of the following holds:

- there are no DiagnosticSecurityLevels referenced in the role of diagnosticSecurityLevel,
- there exists one DiagnosticSecurityLevel referenced in the role of diagnosticSecurityLevel with id matching the given security level.

⌋(*SRS_Diag_04005*)

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_04120*)

[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_04120*)

7.2.7.3 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_04196)*

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 7.2.3.

[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 of a conversation, it shall update the field activityStatus of provided service DiagnosticConversation (see Service Interfaces - Diagnostic Conversation) accordingly. ⌋*(SRS_Diag_04208)*

7.2.7.4 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 DiagnosticResponseToEcuReset-Enum.respondBeforeReset. ⌋*(SRS_Diag_04019)*

[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 DiagnosticEcuReset-

Class.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 7.2.4.2). ⌋(*SRS_Diag_04196*)

7.2.7.5 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, SRS_Diag_04196*)

[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, SRS_Diag_04117*)

[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, SRS_Diag_04117*)

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

The DTC clearing behavior is described in detail in chapter 7.3.4.6. 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). ⌋(*SRS_Diag_04196*)

[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. ⌋(*SRS_Diag_04180*)

The definition of a failure of the non-volatile memory is hardware and project specific. In general if the clear DTC operation could not delete the snapshot records, extended data records and if it could not reset the 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_04117)*

[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_04117)*

[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_04117)*

[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, SRS_Diag_04196)*

[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, SRS_Diag_04196)*

[SWS_DM_00163] Definition of a failed clear operation with event clear allowed and event combination ⌈ If it is requested to clear a single DTC and multiple `DiagnosticEventToTroubleCodeUdsMapping` referencing this `DiagnosticEventToTroubleCodeUdsMapping.troubleCodeUds` the 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)*

7.2.7.5.1 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. ⌋*(SRS_Diag_04197)*

[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 7.3.4.6.

[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)*

7.2.7.6 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

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

7.2.7.6.1 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

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

7.2.7.6.2 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.7.6.3 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.7.6.4 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.7.6.5 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_04180, SRS_Diag_04157)*

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

7.2.7.6.6 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.7.6.7 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.7.6.8 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.7.6.9 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_04180, SRS_Diag_04157, SRS_Diag_04067)*

7.2.7.7 Service 0x22 – ReadDataByIdentifier

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

[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_00412] Check requested number of DataIdentifiers ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), the DM shall check the number of

the requested DataIdentifiers against the configuration parameter maxDidToRead. ⌋ *(SRS_Diag_04203)*

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

[SWS_DM_00413] Check supported DataIdentifier in active session ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), a requested DataIdentifier shall be considered as supported in active session if and only if the DataIdentifier is supported according to [SWS_DM_00409] and the active session passes the execution permission check as per [SWS_DM_00410]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00414] Check supported DataIdentifier on active security level ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), a requested DataIdentifier shall be considered as supported on active security level if and only if the DataIdentifier is supported according to [SWS_DM_00409] and the active security level passes the execution permission check as per [SWS_DM_00411]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00408] Retrieving data for requested DataIdentifier ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), the DM shall retrieve the data for a DataIdentifier according to its configuration as described in [SWS_DM_00401], [SWS_DM_00402], [SWS_DM_00403], [SWS_DM_00404]. ⌋*(SRS_Diag_04097)*

[SWS_DM_00177] Reaction on ApplicationError ⌈ On reception of the UDS Service ReadDataByIdentifier (0x22), if DM requests data via an AA and this AA raises an ApplicationError typed as UDSServiceFailed, then DM shall abort service processing and return a negative response with the value of the errorContext of type UDSResponseCodeType associated to UDSServiceFailed. ⌋*(SRS_Diag_04196)*

Note: If multiple DataIdentifer are requested within one ReadDataByIdentifier request, [SWS_DM_00177] might result in a deviation from ISO 14229-1[1] in case the AA raises an ApplicationError of type UDSServiceFailed with errorContext set to NRC 0x31. According to ISO 14229-1[1], chapter 10.2, a tester expects to receive NRC 0x31 only in case **none** of the requested DataIdentifier are supported. Handling of ApplicationErrors as described in [SWS_DM_00177] might lead to NRC 0x31 on processing one of the requested DataIdentfier without checking the other requested DataIdentifier.

7.2.7.8 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, SRS_Diag_04005)*

[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_04203)*

[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_04203)*

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

[SWS_DM_00250] Notification about security-level change ⌈ If DM did successfully change the security-level of a conversation, it shall update the diagnosticSession of Status field of provided service DiagnosticConversation (see Service Interfaces - DiagnosticConversation) accordingly. Whether a security level is applicable by the DiagnosticSecurityAccess is defined by securityLevel. ⌋ *(SRS_Diag_04208)*

[SWS_DM_00270] Counting of attempts to change security level ⌈ The 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. ⌋ *(SRS_Diag_04208)*

[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. ⌋ *(SRS_Diag_04208)*

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

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

tExpired).
⌋(*SRS_Diag_04208*)

7.2.7.9 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*)

7.2.7.10 Service 0x2E – WriteDataByIdentifier

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

[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_00415] Check supported DataIdentifier ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported if and only if there exists a DiagnosticDataIdentifier with id matching the DataIdentifier and this DiagnosticDataIdentifier is referenced by a DiagnosticWriteDataByIdentifier. ⌋(*SRS_Diag_04203*)

[SWS_DM_00416] Check supported DataIdentifier in active session ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported in active session if and only if the DataIdentifier is supported according to [SWS_DM_00415] and the active session passes the execution permission check as per [SWS_DM_00410]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00417] Check supported DataIdentifier on active security level ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported on active security level if and only if the DataIdentifier is supported according to [SWS_DM_00415] and the active security level passes the execution permission check as per [SWS_DM_00411]. ⌋*(SRS_Diag_04203)*

[SWS_DM_00418] Writing data for requested DataIdentifier ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), the DM shall write the data for a DataIdentifier according to its configuration as described in [SWS_DM_00405], [SWS_DM_00406], [SWS_DM_00407]. ⌋*(SRS_Diag_04097)*

[SWS_DM_00419] Reaction on ApplicationError ⌈ On reception of the UDS Service WriteDataByIdentifier (0x2E), if DM processes data via an AA and this AA raises an ApplicationError typed as UDSServiceFailed, then DM shall abort service processing and return a negative response with the value of the errorContext of type UDSResponseCodeType associated to UDSServiceFailed. ⌋*(SRS_Diag_04196)*

7.2.7.11 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_04203)*

[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_04203)*

[SWS_DM_00210] RoutineControl startRoutine processing ⌈ The DM shall call the method Start of the interface RoutineService (see 8.2.1.6) 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 8.2.1.6) 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 8.2.1.6) to process the subfunction stopRoutine. ⌋*(SRS_Diag_04196)*

7.2.7.12 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, SRS_Diag_04033)*

[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)*

7.2.7.13 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_04033)*

7.2.7.14 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_04033)*

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_04033)*

7.2.7.15 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_04033)*

[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_04033)*

7.2.7.16 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)*

7.2.7.17 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, SRS_Diag_04159)*

[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_04203)*

[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_04203, SRS_Diag_04115)*

[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, SRS_Diag_04159)*

[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, SRS_Diag_04159)*

7.3 Event memory management

7.3.1 Diagnostic Events

7.3.1.1 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.

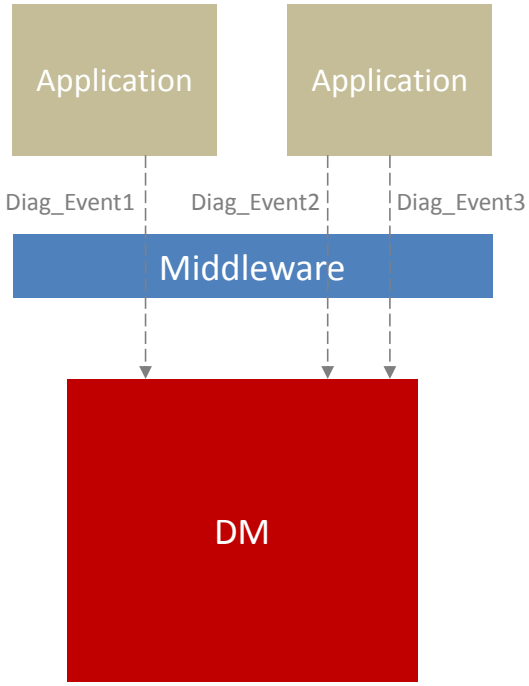

Figure 7.2: Example of diagnostic event usage

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

[SWS_DM_00008] Diagnostic event processing interface ⌈ The DM shall provide a service interface DiagnosticEvent (see Service Interfaces - 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)

7.3.1.2 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 method of the DiagnosticMonitor 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. ⌋*(SRS_Diag_04186)*

The detailed description of enable conditions can be found in 7.3.4.3 chapter.

[SWS_DM_00070] Monitor initialization for DTC setting re-enabling reason ⌈ The DM shall publish the InitMonitor event with REENABLED value, in case DTC setting is re-enabled via the UDS service request ControlDTCSetting - 0x85 (see ISO 14229-1[1]). ⌋*(SRS_Diag_04186)*

[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

⌋(*SRS_Diag_04186*)

The detailed description of storage conditions can be found in 7.3.4.4 chapter.

7.3.1.3 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 DiagnosticMonitor service interfaces ⌈ The DM shall provide a service interface DiagnosticMonitor (see Service Interfaces - DiagnosticMonitor) per configured DiagnosticEvent. ⌋(*SRS_Diag_04179*)

[SWS_DM_00166] Trigger to process event status ⌈ If MonitorAction of service interface DiagnosticMonitor 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 DiagnosticMonitor shall be discarded, if the referenced DiagnosticOperationCycle of this DiagnosticEvent (via DiagnosticEventToOperationCycleMapping) is set to END. ⌋(*SRS_Diag_04178*)

7.3.1.4 Debouncing

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

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

- Counter-based debouncing
- Time-based debouncing

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

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_04068)*

7.3.1.4.1 Counter-based debouncing

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

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

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

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

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

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

[SWS_DM_00020] Internal debounce counter decrementation ⌈ The DM shall decrement the event's internal debounce counter by the configured step-size value of DiagEventDebounceCounterBased.counterDecrementStepSize, when the monitor reports PREPASSED. ⌋*(SRS_Diag_04125)*

[SWS_DM_00025] Qualified passed event using counter-based debouncing ⌈ If the internal debounce counter is less or equal to DiagEventDebounceCounter-

Based.counterPassedThreshold the DM shall process the event as PASSED. ⌋ *(SRS_Diag_04125)*

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

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

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

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

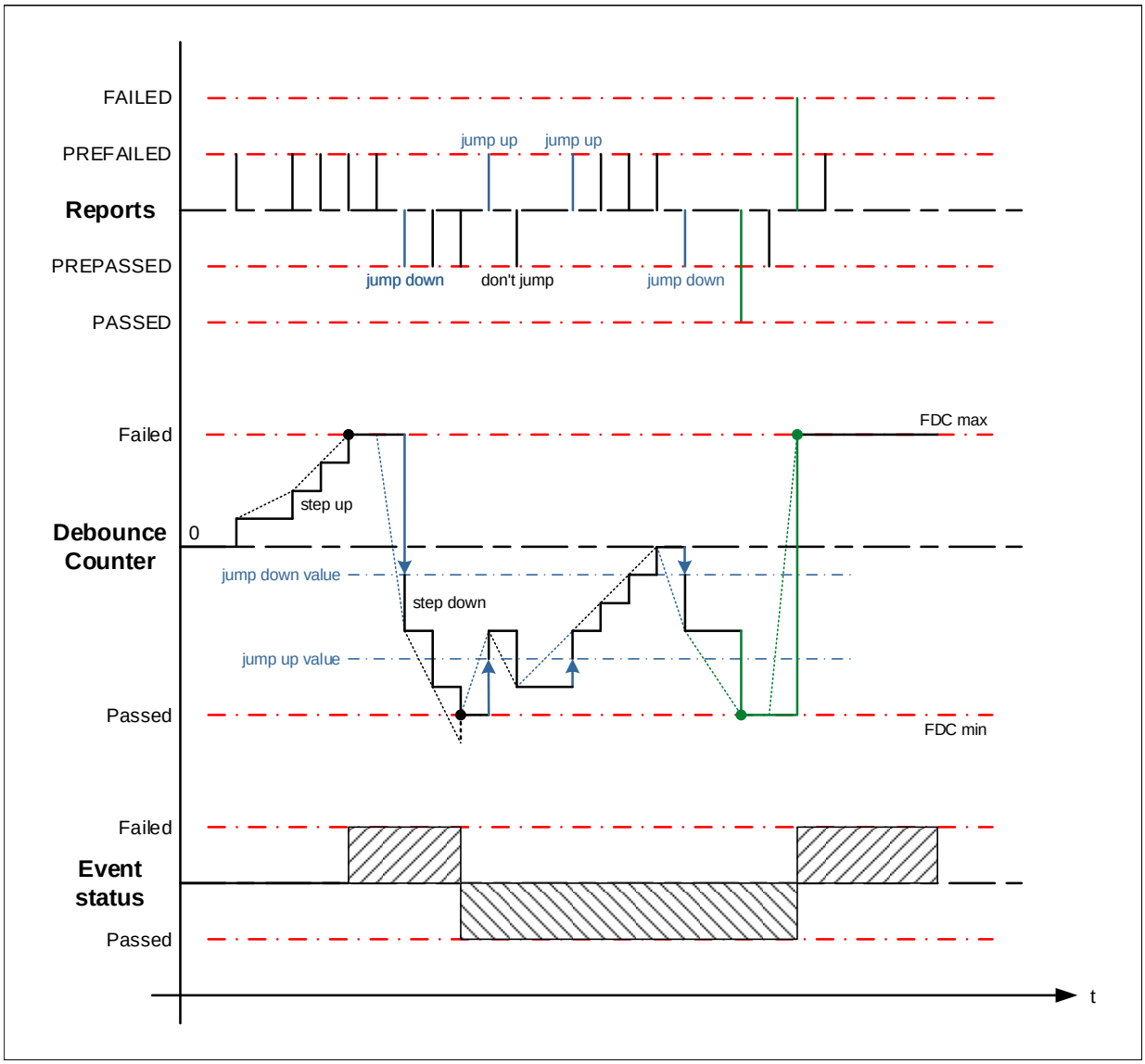

Figure 7.3: Counter-based debouncing

[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_04124)*

7.3.1.4.2 Time-based debouncing

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

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

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

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

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

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

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

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

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

⌋*(SRS_Diag_04068)*

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

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

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

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

⌋*(SRS_Diag_04068)*

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

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

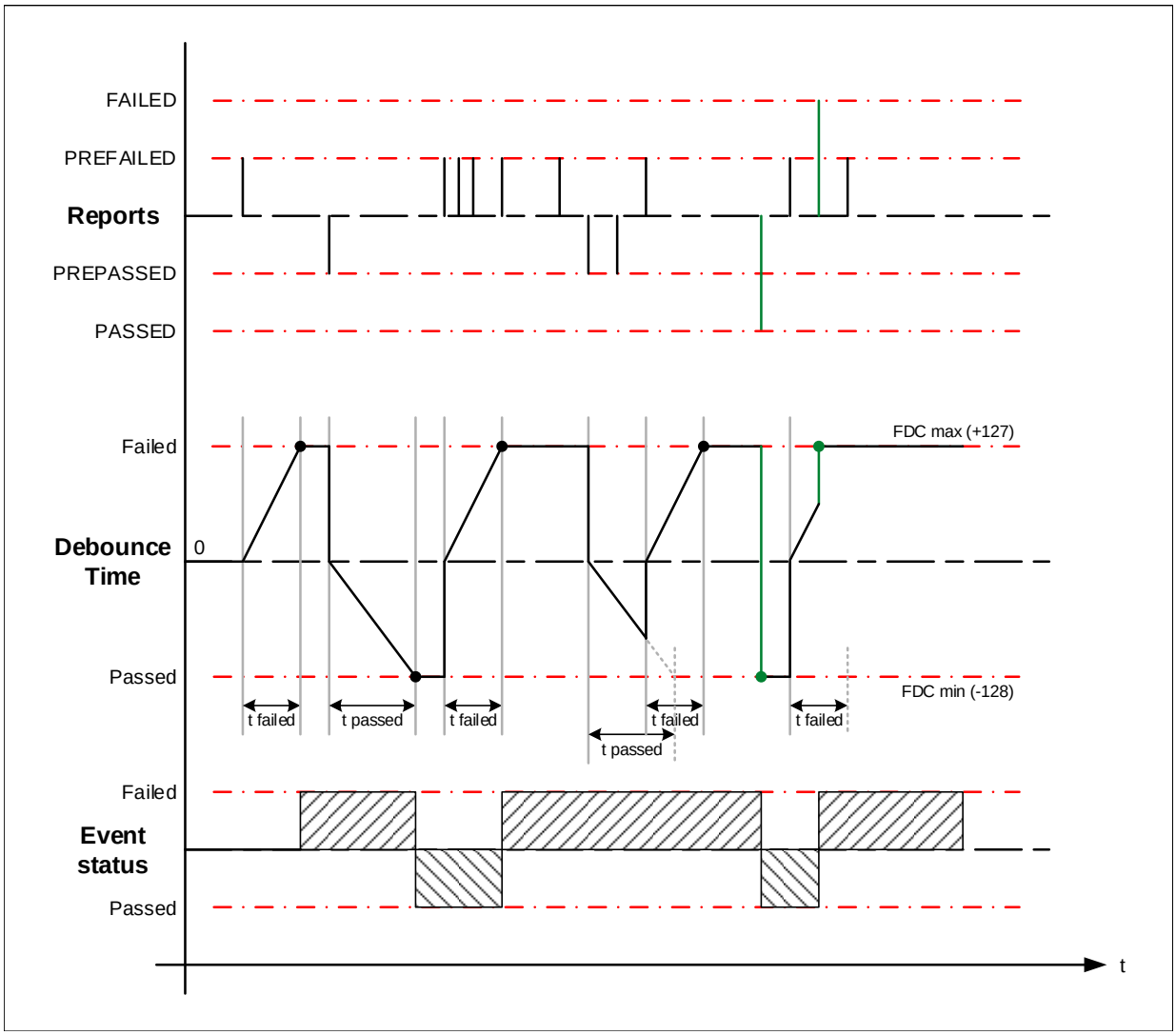

Figure 7.4: Timer based debouncing

[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_04068)*

7.3.1.4.3 Debounce algorithm reset

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

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

[SWS_DM_00026] Application resetting the debounce counter ⌈ If MonitorAction of service interface DiagnosticMonitor is called with RESET_DEBOUNCING, the DM shall reset the debounce counter. ⌋*(SRS_Diag_04068)*

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

[SWS_DM_00037] Debounce time freeze request ⌈ If MonitorAction of service interface DiagnosticMonitor is called with FREEZE_DEBOUNCING, the DM shall freeze the related debounce timer for event with configured timer based debouncing. ⌋ *(SRS_Diag_04068)*

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

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

7.3.1.4.4 Dependencies to enable conditions

As described in chapter 7.3.4.3 enable conditions are used to suppress the result of reported event status information. Enable conditions have also effect on the debouncing behavior of the DM.

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

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

7.3.1.4.5 Dependencies to UDS service 0x85 ControlDTCSettings

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

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

7.3.2 DTC Status processing

7.3.2.1 Status processing

[SWS_DM_00213] DTC status processing ⌈

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

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

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

Note that if configured, `PREPASSED` or `PREFAILED` status reports reported via `MonitorAction` trigger debounce mechanisms (see 7.3.1.4). 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_04151)*

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

[SWS_DM_00216] DTC status bit transitions triggered by operation cycle changes ⌈ The `DM` shall process the UDS status byte triggered by operation cy-

cle 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_04180, SRS_Diag_04157, SRS_Diag_04067)

[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_04180, SRS_Diag_04157, SRS_Diag_04067)

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

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

If there is an indicator mapped to the DTC, the 'warningIndicatorRequested' bit is handled as described in 7.3.2.3.

7.3.2.2 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 EventStatus field of the corresponding DiagnosticEvent service interface and via the GetCurrentStatus method 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 EventStatus field of the corresponding DiagnosticEvent service interface and via the GetCurrentStatus method and DTCStatusChanged event of the corresponding DTCInformation service interface. ⌋(SRS_Diag_04183)

7.3.2.3 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_04204)*

For confirmation check [SWS_DM_00218].

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

7.3.3 Operation Cycles Management

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

Examples of typical operation cycles are:

- Ignition on/off cycles
- Power up/power down cycle
- Accumulated operating time cycles

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

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

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

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

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

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

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

7.3.4 Event memory

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

There can be multiple event memories handled by the DM.

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

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

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

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

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

7.3.4.1 DTC Introduction

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

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

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

7.3.4.1.1 Format

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

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

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

- iso11992_4
- iso14229_1
- saeJ2012_da

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

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

- iso15031_6
- saeJ1939_73

⌋*(SRS_Diag_04201)*

7.3.4.1.2 Groups

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

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

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

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

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

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

7.3.4.2 Destination

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

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

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

7.3.4.3 EnableConditions

DiagnosticEnableConditions are mapped to DiagnosticEvents by DiagnosticEventToEnableConditionGroupMappings.

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

7.3.4.4 StorageConditions

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

DiagnosticStorageConditions are mapped to DiagnosticEvents by DiagnosticEventToStorageConditionGroupMappings.

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

7.3.4.5 DTC related data

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

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

⌋*(SRS_Diag_04211)*

7.3.4.5.1 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.

7.3.4.5.2 Storage of snapshot record data

[SWS_DM_00151] Snapshot record numeration ⌈ In case DiagnosticCommonProps.typeOfFreezeFrameRecordNumeration is set to calculated, snapshot records shall be numbered consecutively starting with 1 in their chronological order.

If the parameter is set to configured, configured record numbers shall be used based on the DiagnosticFreezeFrame.recordNumber configuration parameters of the respective snapshot records. ⌋(*SRS_Diag_04205*, *SRS_Diag_04189*)

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

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

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

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

7.3.4.5.3 Storage of extended data

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

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

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

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

7.3.4.6 Clearing DTCs

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

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

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

7.3.4.6.1 Locking of the DTC clearing process by an client

The DM supports more than one diagnostic clients as described in chapter 7.2.3. 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_04117)*

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

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

[SWS_DM_00147] Behavior while trying to clear DTCs on a locked DiagnosticMemoryDestination ⌈ If the DM is requested to clear DTCs of a DiagnosticMemoryDestination and the DM has locked this DiagnosticMemoryDestination for clearing DTCs according to [SWS_DM_00144], the DM shall refuse the

second clear DTC operation and shall return a NRC 0x22 (ConditionsNotCorrect). ⌋ *(SRS_Diag_04117)*

7.3.4.6.2 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_04117)*

[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_04117)*

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_04117)*

[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_04117)*

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

[SWS_DM_00123] Block status byte clearing during a clear DTC operation ⌈ If the DM is requested to clear a DTC 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_04117)*

[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_04117*)

[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_04117*)

7.3.4.6.3 DTC clearing triggered by application

Besides the UDS request ClearDiagnosticInformation according to 7.2.7.5.1 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 7.2.7.5.1. This could be realized using a dedicated diagnostic routine service, whose application is in charge of the clearing process.

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

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

[SWS_DM_00261] Usage of ClearDTC Interface ⌈ If the method Clear of the 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 associated to the instance of the clearDTC interface only. ⌋(*SRS_Diag_04194*)

[SWS_DM_00263] ClearDTC call on invalid DTC or DTCgroup ⌈ If the method Clear of the 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_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, 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)*

7.3.4.7 Aging

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

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

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

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

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

[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_04133)*

[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_04133)*

[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_04140)*

7.4 Required Configuration

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

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

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

[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. ⌋ *(SRS_Eth_00026)*

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.

7.5 Diagnostic Data Management

In various situations, the DM facilitates reading or writing of particular diagnostic data. One needs to distinguish between internal and external diagnostic data. By definition, internal data is managed by the DM itself, and external data is managed by external applications. In the latter case, communication between DM and the external applica-

tion takes place via Service Interfaces. There are several Service Interfaces defined concerning diagnostic data.

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

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

7.5.1 Internal and External Diagnostic Data Elements

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

Table 7.1 gives a list of the supported internal DiagnosticDataElements, where

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

Content describes the actual content of the data,

Format describes the data format of the DiagnosticDataElement.

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

Data Provider	Content	Format	Context
DEM_AGINGCTR_DOWNCNT	Down-counting aging counter of contextual DTC	1 byte	DEM
DEM_AGINGCTR_UPCNT	Up-counting aging counter of contextual DTC	1 byte	DEM
DEM_AGINGCTR_UPCNT_FIRST_ACTIVE	Up-counting aging counter of contextual DTC, starting at 1 when aging starts	1 byte	DEM
DEM_CURRENT_FDC	Fault Detection Counter of contextual DTC	1 byte	DEM
DEM_CYCLES_SINCE_FIRST_FAILED	Operation Cycle Counter of contextual DTC– Cycles since first failed	1 byte	DEM
DEM_CYCLES_SINCE_LAST_FAILED	Operation Cycle Counter of contextual DTC – Cycles since last failed	1 byte	DEM
DEM_FAILED_CYCLES	Operation Cycle Counter of contextual DTC – Failed cycles	1 byte	DEM
DEM_MAX_FDC_DURING_CURRENT_CYCLE	Fault Detection Counter maximum value during current operation cycle of contextual DTC	1 byte	DEM
DEM_MAX_FDC_SINCE_LAST_CLEAR	Fault Detection Counter maximum value since last clear of contextual DTC	1 byte	DEM
DEM_OCCCTR	Occurrence counter of contextual DTC	1 byte	DEM
DEM_OVFLIND	Overflow indication of contextual DTC (0 = False, 1 = True)	1 byte	DEM

DEM_SIGNIFICANCE	Event significance of contextual DTC (refer to DemDTCSignificance) (0 = OCCURRENCE, 1 = FAULT)	1 byte	DEM
DEM_PRIORITY	Priority of the contextual DTC	1 byte	DEM
DCM_SESSION	Current session of contextual Diagnostic Protocol	1 byte	DCM
DCM_SECURITY_LEVEL	Current security level of contextual Diagnostic Protocol	1 byte	DCM

Table 7.1: Supported internal DiagnosticDataElements

[SWS_DM_00393] Retrieving data for internal DiagnosticDataElements ⌈ If DM requires to provide or store data configured as internal DiagnosticDataElement which is supported by DM according to Table 7.1, then DM shall use the respective internally managed data value as defined in Table 7.1. ⌋*(SRS_Diag_04097)*

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

An internal DiagnosticDataElement is called DCM-exclusive resp. DEM-exclusive if the context of the name token described in Table 7.1 is set accordingly. The implicit restriction of such DiagnosticDataElements to the context in which they are defined is made explicit in the following requirements. These requirements are formulated in a way that Table 7.1 might in future be extended by internal DiagnosticDataElements not restricted to exclusive use within a DCM resp. DEM context.

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

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

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

[SWS_DM_00397] Retrieving data for external DiagnosticDataElements ⌈ If DM is required to read data configured as external DiagnosticDataElement, then DM shall utilize the associated RPortPrototype typed by the DataElement Service Interface as specified in section 8.2.2.9 and call its Read method. ⌋*(SRS_Diag_04097)*

Note: In general, there are multiple instances of DataElement Service Interface available in the running system. Which instance to choose for the given request to read an external DiagnosticDataElement is part of system integration. Support for this integration is provided by DiagnosticMappings described in section 7.5.2.1.

7.5.2 Reading and Writing Diagnostic Data Identifier

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

- reading each DiagnosticDataElement contained in the DiagnosticDataIdentifier independently as described in section 7.5.1,
- reading or writing the DiagnosticDataIdentifier as a whole via the DataIdentifier service interface as specified in section 8.2.1.5,
- reading or writing the DiagnosticDataIdentifier as a whole via the GenericUDSService service interface as specified in section 8.2.1.3.

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

7.5.2.1 Supported Diagnostic Mappings

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

DiagnosticMapping	**diagnostics endpoint**	**target endpoint**
DiagnosticDemProvidedDataMapping	DiagnosticDataElement	DM internal data provider
DiagnosticServiceDataMapping	DiagnosticDataElement	DataPrototype
DiagnosticServiceSwMapping	DiagnosticDataElement or DiagnosticServiceInstance with (if applicable) reference to DiagnosticDataIdentifier	SwcServiceDependency

Table 7.2: Diagnostic Mappings

The DiagnosticDemProvidedDataMapping is used to distinguish between internal and external DiagnosticDataElement as described in section 7.5.1.

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

Concerning the DiagnosticServiceSwMapping, the DM configuration supports two kinds of mappings related to DiagnosticDataElement and DiagnosticDataIdentifier:

Mapping for DiagnosticDataElements:
DiagnosticServiceSwMapping maps a DiagnosticDataElement in the role of diagnosticDataElement to a SwcServiceDependency in the role of mappedSwcServiceDependencyInExecutable.

Mapping for DiagnosticDataIdentifier:
DiagnosticServiceSwMapping maps a DiagnosticDataByIdentifier in the role of serviceInstance (with reference to a DiagnosticAbstractDataIdentifier in the role of dataIdentifier) to a SwcServiceDependency in the role of mappedSwcServiceDependencyInExecutable.

For the second kind of mapping, DiagnosticDataByIdentifier is realized either by a DiagnosticReadDataByIdentifier or by a DiagnosticWriteDataByIdentifier, each with reference to an explicitly given DiagnosticDataIdentifier.

Details regarding the modeling of diagnostic mappings can be found in the TPS Manifest Specification [9].

7.5.2.2 Reading Diagnostic Data Identifier

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

[SWS_DM_00402] Reading Diagnostic Data Identifier by DataIdentifier interface ⌈ If DM is required to read data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticReadDataByIdentifier service and this service is referenced by a DiagnosticServiceSwMapping of category DATA_IDENTIFIER, then DM shall use the DataIdentifier interface associated to the DiagnosticDataIdentifier as defined in section 8.2.1.5 for reading the data. ⌋ *(SRS_Diag_04097)*

[SWS_DM_00403] Reading Diagnostic Data Identifier by GenericUDSService interface ⌈ If DM is required to read data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticReadDataByIdentifier service and this service is referenced by a DiagnosticServiceSwMapping of category GENERIC_UDS_SERVICE, then DM shall use the instance of the GenericUDSService interface referenced by the DiagnosticServiceSwMapping and call its Service method with SID set to 0x22 and requestData set to the id of the DiagnosticDataIdentifier. ⌋*(SRS_Diag_04097)*

[SWS_DM_00404] Default Service Interface for reading `DiagnosticDataIdentifier` ⌈ If DM is required to read data configured as DiagnosticDataIdentifier and none of the requirements [SWS_DM_00401], [SWS_DM_00402], [SWS_DM_00403] applies, then DM shall utilize the associated RPortPrototype typed by the DataIdentifier Service Interface as specified in section 8.2.1.5 and call its Read method. ⌋*(SRS_Diag_04097)*

Note: The default configuration as described in [SWS_DM_00404] assumes that there is a single instance of PPortPrototype defined in the system matching the RPortPrototype associated to the requested DiagnosticDataIdentifier as defined in section 8.2.1.5. In this case, it is part of integration step to link these two ports.

7.5.2.3 Writing Diagnostic Data Identifier

[SWS_DM_00405] Writing Diagnostic Data Identifier by DataIdentifier interface ⌈ If DM is required to write data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticWriteDataByIdentifier service and this service is referenced by a DiagnosticServiceSwMapping of category DATA_IDENTIFIER, then DM shall use the DataIdentifier interface associated to the DiagnosticDataIdentifier as defined in section 8.2.1.5 for writing the data. ⌋ *(SRS_Diag_04097)*

[SWS_DM_00406] Writing Diagnostic Data Identifier by GenericUDSService interface ⌈ If DM is required to writing data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticWriteDataByIdentifier service and this service is referenced by a DiagnosticServiceSwMapping of category GENERIC_UDS_SERVICE, then DM shall use the instance of the GenericUDSService interface referenced by the DiagnosticServiceSwMapping and call its Service method with SID set to 0x2E and requestData set to the id of this DiagnosticDataIdentifier followed by the data to be written to this DiagnosticDataIdentifier. ⌋*(SRS_Diag_04097)*

[SWS_DM_00407] Default Service Interface for writing `DiagnosticDataIdentifier` ⌈ If DM is required to write data configured as DiagnosticDataIdentifier and none of the requirements [SWS_DM_00405], [SWS_DM_00406] applies, then DM shall utilize the associated RPortPrototype typed by the DataIdentifier Service Interface as specified in section 8.2.1.5 and call its Write method. ⌋ *(SRS_Diag_04097)*

Note: The default configuration as described in [SWS_DM_00407] assumes that there is a single instance of PPortPrototype defined in the system matching the RPortPrototype associated to the requested DiagnosticDataIdentifier as defined in section 8.2.1.5. In this case, it is part of integration step to link these two ports.

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 DiagnosticConversationStatusType

Name	DiagnosticConversationStatusType
Kind	STRUCTURE
Subelements	activityStatus ActivityStatusType diagnosticSession DiagnosticSessionType diagnosticSecurityLevel DiagnosticSecurityLevelType
Derived from	-
Description	Represents the status of an active conversation.

Table 8.1: Implementation Data Type - DiagnosticConversationStatusType

8.1.1.2 ActivityStatusType

Name	ActivityStatusType	
Kind	TYPE	
Derived from	uint8	
Description	Type used in TBD	
Range / Symbol	***Limit***	***Description***
kActive	0x00	Currently active; i.e. request is curretnly processed or non-default session is active.
kInactive	0x01	Currently not active

Table 8.2: Implementation Data Type - ActivityStatusType

8.1.1.3 DiagnosticSessionType

Name	DiagnosticSessionType
Kind	STRING
Derived from	-
Description	Represents the Diagnostic Session.

Table 8.3: Implementation Data Type - DiagnosticSessionType

8.1.1.4 DiagnosticSecurityLevelType

Name	DiagnosticSecurityLevelType
Kind	STRING

Derived from	-
Description	Represents the Diagnostic Security Level.

Table 8.4: Implementation Data Type - DiagnosticSecurityLevelType

8.1.1.5 DiagnosticConversationIdentifierType

Name	DiagnosticConversationIdentifierType
Kind	STRUCTURE
Subelements	diagnosticProtocolKind targetAddress `UdsAddressType` sourceAddress `UdsAddressType`
Derived from	-
Description	Characterizes an ongoing Diagnostic Conversation.

Table 8.5: Implementation Data Type - DiagnosticConversationIdentifierType

8.1.1.6 UdsAddressType

Name	UdsAddressType
Kind	TYPE
Derived from	uint16
Description	Represents the UDS Address as defined in ISO-14229-1.

Table 8.6: Implementation Data Type - UdsAddressType

8.1.1.7 ByteVectorType

Name	ByteVectorType
Kind	VECTOR
Subelements	byte `uint8`
Derived from	-
Description	DataArrayType

Table 8.7: Implementation Data Type - ByteVectorType

8.1.1.8 MetaInfoKeyType

Name	MetaInfoKeyType	
Kind	STRING	
Derived from	-	
Description	Represents the predefined/valid keys, which are available within the optional MetaInfo the DM provides in service processor calls.	
Range / Symbol	***Limit***	***Description***
kSA	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"

kTA	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"
kRequestHandle	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.
kRequestType	0x03	Indicator whether request is functional or physical addressed. The value in the MetaInf Map for this key, will be either "PHYS" or "FUNC"
kSuppPosResponse	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"
kProtocolId	0x07	The identifier which can be used to call FindService(PROTOCOL_ID) to find the DiagnosticConversation ServiceInstance.
kDoIPLocalIP	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
kDoIPLocalPort	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.
kDoIPRemoteIP	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)
kDoIPRemotePort	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.

Table 8.8: Implementation Data Type - MetaInfoKeyType

8.1.1.9 MetaInfoType

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

Table 8.9: Implementation Data Type - MetaInfoType

8.1.1.10 MetaInfoValueType

Name	MetaInfoValueType
Kind	STRING
Derived from	-

Description	

Table 8.10: Implementation Data Type - MetaInfoValueType

8.1.1.11 KeyCompareResultType

Name	KeyCompareResultType	
Kind	TYPE	
Derived from	uint8	
Description		
Range / Symbol	***Limit***	***Description***
kKeyValid	0x00	Key is valid
kKeyInvalid	0x01	Key is invalid

Table 8.11: Implementation Data Type - KeyCompareResultType

8.1.1.12 ControlDtcStatusType

Name	ControlDtcStatusType	
Kind	TYPE	
Derived from	uint8	
Description	Type for ControlDTCStatus status	
Range / Symbol	***Limit***	***Description***
kDTCSettingOn	0x00	Updating of diagnostic trouble code status bits is under normal operating conditions
kDTCSettingOff	0x01	Updating of diagnostic trouble code status bits is stopped

Table 8.12: Implementation Data Type - ControlDtcStatusType

8.1.1.13 CommunicationControlStatusType

Name	CommunicationControlStatusType
Kind	TYPE
Derived from	uint8
Description	CommunicationControlStatusType

Table 8.13: Implementation Data Type - CommunicationControlStatusType

8.1.1.14 ConfirmationStatusType

Name	ConfirmationStatusType	
Kind	TYPE	
Derived from	uint8	
Description	Type used in the method confirmed for the status of the service processing	
Range / Symbol	***Limit***	***Description***
kResPosOk	0x00	Positive response has been sent out successfully
kResPosNotOk	0x01	Positive response has not been sent out successfully
kResNegOk	0x02	Negative response has been sent out successfull

kResNegNotOk	0x03	Negative response has not been sent out successfully
kResPosSuppressed	0x04	Positive answer suppressed

Table 8.14: Implementation Data Type - ConfirmationStatusType

8.1.1.15 StateType

Name	StateType
Kind	TYPE
Derived from	boolean
Description	boolean

Table 8.15: Implementation Data Type - StateType

8.1.1.16 SIDType

Name	SIDType
Kind	TYPE
Derived from	uint8
Description	SIDType

Table 8.16: Implementation Data Type - SIDType

8.1.1.17 ClearFailedReasonType

Name	ClearFailedReasonType	
Kind	TYPE	
Derived from	uint8	
Description	ClearFailedReasonType	
Range / Symbol	***Limit***	***Description***
kFailed	0x00	Failed to clear the DTC due to any other reason
kBusy	0x01	DTC not cleared, as another clearing process is in progress. The caller can retry later
kMemoryError	0x02	An error occurred during erasing a memory location
kWrongDtc	0x03	DTC value does not exist in the current configuration

Table 8.17: Implementation Data Type - ClearFailedReasonType

8.1.1.18 UDSResponseCodeType

Name	UDSResponseCodeType	
Kind	TYPE	
Derived from	uint8	
Description	UDSResponseCodeType	
Range / Symbol	***Limit***	***Description***
kGeneralReject	0x10	According to ISO
kServiceNotSupported	0x11	According to ISO
kSubfunctionNotSupported	0x12	According to ISO

kIncorrectMessageLengthOr InvalidFormat	0x13	According to ISO
kBusyRepeatRequest	0x21	According to ISO
kConditionsNotCorrect	0x22	According to ISO
kRequestSequenceError	0x24	According to ISO
kNoResponseFromSubnet Component	0x25	According to ISO
kFailurePreventsExecutionOf RequestedAction	0x26	According to ISO
kRequestOutOfRange	0x31	According to ISO
kSecurityAccessDenied	0x33	According to ISO
kInvalidKey	0x35	According to ISO
kExceedNumberOfAttempts	0x36	According to ISO
kRequiredTimeDelayNot Expired	0x37	According to ISO
kUploadDownloadNotAccepted	0x70	According to ISO
kTransferDataSuspended	0x71	According to ISO
kGeneralProgrammingFailure	0x72	According to ISO
kWrongBlockSequenceCounter	0x73	According to ISO
kRequestCorrectlyReceived ResponsePending	0x78	According to ISO
kSubFunctionNotSupportedIn ActiveSession	0x7E	According to ISO
kServiceNotSupportedInActive Session	0x7F	According to ISO
kNoProcessingNoResponse	0xFF	Deviating from ISO - no further service processing and no response (silently ignore request message).

Table 8.18: Implementation Data Type - UDSResponseCodeType

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 / Symbol	***Limit***	***Description***
kPassed	0x00	Monitor reports qualified test result passed.
kFailed	0x01	Monitor reports qualified test result failed
kPrepassed	0x02	Monitor reports unqualified test result pre-passed.
kPrefailed	0x03	Monitor reports unqualified test result pre-failed.
kFdcThresholdReached	0x04	Monitor triggers the storage of ExtendedDataRecords and Freeze Frames (if the triggering condition is connected to this threshold).
kResetTestFailed	0x05	Reset TestFailed Bit without any other side effects like readiness
kFreezeDebouncing	0x06	Freeze the internal debounce counter/timer.
kResetDebouncing	0x07	Reset the internal debounce counter/timer.
kPrestore	0x08	Capture and prestores the freeze frame data.

kClearPrestore	0x09	Clears a prestored freeze frame.

Table 8.19: Implementation Data Type - MonitorActionType

8.1.2.2 DebouncingStateType

Name	DebouncingStateType			
Kind	TYPE			
Derived from	uint8			
Description	DebouncingStateType			
Name	***Limit***	***Mask***	***State***	***Description***
kTemporarilyDefective	0	0x01	FALSE	Bit 0: Temporarily Defective (corresponds to 0 < FDC < 127)
kTemporarilyDefective	1	0x01	TRUE	Bit 0: Temporarily Defective (corresponds to 0 < FDC < 127)
kFinallyDefective	0	0x02	FALSE	Bit 1: finally Defective (corresponds to FDC = 127)
kFinallyDefective	2	0x02	TRUE	Bit 1: finally Defective (corresponds to FDC = 127)
kTemporarilyHealed	0	0x04	FALSE	Bit 2: temporarily healed (corresponds to -128 < FDC < 0)
kTemporarilyHealed	4	0x04	TRUE	Bit 2: temporarily healed (corresponds to -128 < FDC < 0)
kTestComplete	0	0x08	FALSE	Bit 3: Test complete (corresponds to FDC = -128 or FDC = 127)
kTestComplete	8	0x08	TRUE	Bit 3: Test complete (corresponds to FDC = -128 or FDC = 127)
kDTRUpdate	0	0x10	FALSE	Bit 4: DTR Update (= Test complete && Debouncing complete && enable conditions / storage conditions fulfilled)
kDTRUpdate	16	0x10	TRUE	Bit 4: DTR Update (= Test complete && Debouncing complete && enable conditions / storage conditions fulfilled)

Table 8.20: Implementation Data Type - DebouncingStateType

8.1.2.3 DTCFormatType

Name	DTCFormatType	
Kind	TYPE	
Derived from	uint8	
Description	DTCFormatType	
Range / Symbol	***Limit***	***Description***
kDTCFormatOBD	0	
kDTCFormatUDS	1	
kDTCFormatJ1939	2	

Table 8.21: Implementation Data Type - DTCFormatType

8.1.2.4 DTCGroupType

Name	DTCGroupType
Kind	TYPE
Derived from	uint32
Description	uint32

Table 8.22: Implementation Data Type - DTCGroupType

8.1.2.5 DTCStatusChangedType

Name	DTCStatusChangedType
Kind	STRUCTURE
Subelements	DTC `DTCType` udsStatusByteOld `UdsStatusByteType` udsStatusByteNew `UdsStatusByteType`
Derived from	-
Description	DTCStatusChangedType

Table 8.23: Implementation Data Type - DTCStatusChangedType

8.1.2.6 DTCType

Name	DTCType
Kind	TYPE
Derived from	uint32
Description	uint32

Table 8.24: Implementation Data Type - DTCType

8.1.2.7 UdsStatusByteType

Name	UdsStatusByteType			
Kind	TYPE			
Derived from	uint8			
Description	UdsStatusByteType			
Name	***Limit***	***Mask***	***State***	***Description***
kUDSStatusTF	0	0x01	FALSE	bit 0: TestFailed
kUDSStatusTF	1	0x01	TRUE	bit 0: TestFailed
kUDSStatusTFTOC	0	0x02	FALSE	bit 1: TestFailedThisOperationCycle
kUDSStatusTFTOC	2	0x02	TRUE	bit 1: TestFailedThisOperationCycle
kUDSStatusPDTC	0	0x04	FALSE	bit 2: PendingDTC
kUDSStatusPDTC	4	0x04	TRUE	bit 2: PendingDTC
kUDSStatusCDTC	0	0x08	FALSE	bit 3: ConfirmedDTC
kUDSStatusCDTC	8	0x08	TRUE	bit 3: ConfirmedDTC
kUDSStatusTNCSLC	0	0x10	FALSE	bit 4: TestNotCompletedSinceLastClear
kUDSStatusTNCSLC	16	0x10	TRUE	bit 4: TestNotCompletedSinceLastClear
kUDSStatusTFSLC	0	0x20	FALSE	bit 5: TestFailedSinceLastClear
kUDSStatusTFSLC	32	0x20	TRUE	bit 5: TestFailedSinceLastClear
kUDSStatusTNCTOC	0	0x40	FALSE	bit 6: TestNotCompletedThisOperationCycle

kUDSStatusTNCTOC	64	0x40	TRUE	bit 6: TestNotCompletedThisOperationCycle
kUDSStatusWIR	0	0x80	FALSE	bit 7: WarningIndicatorRequested
kUDSStatusWIR	128	0x80	TRUE	bit 7: WarningIndicatorRequested

Table 8.25: Implementation Data Type - UdsStatusByteType

8.1.2.8 EventStatusByteType

Name	EventStatusByteType			
Kind	TYPE			
Derived from	uint8			
Description	EventStatusByteType			
Name	***Limit***	***Mask***	***State***	***Description***
kUDSStatusTF	0	0x01	FALSE	bit 0: TestFailed
kUDSStatusTF	1	0x01	TRUE	bit 0: TestFailed
kUDSStatusTFTOC	0	0x02	FALSE	bit 1: TestFailedThisOperationCycle
kUDSStatusTFTOC	2	0x02	TRUE	bit 1: TestFailedThisOperationCycle
kUDSStatusTNCTOC	0	0x40	FALSE	bit 6: TestNotCompletedThisOperationCycle
kUDSStatusTNCTOC	64	0x40	TRUE	bit 6: TestNotCompletedThisOperationCycle

Table 8.26: Implementation Data Type - EventStatusByteType

8.1.2.9 FaultDetectionCounterType

Name	FaultDetectionCounterType
Kind	TYPE
Derived from	sint8
Description	sint8

Table 8.27: Implementation Data Type - FaultDetectionCounterType

8.1.2.10 IndicatorStatusTyp

Name	IndicatorStatusType	
Kind	TYPE	
Derived from	uint8	
Description	IndicatorStatusType	
Range / Symbol	***Limit***	***Description***
kOff	0x00	Indicator off mode
kContinous	0x01	Indicator continuously on mode
kBlinking	0x02	Indicator blinking mode
kBlinkingAndContinous	0x03	Indicator blinking or continuously on mode
kSlowFlash	0x04	Indicator slow flashing mode
kFastFlash	0x05	Indicator fast flashing mode
kOnDemand	0x06	Indicator on-demand mode
kShort	0x07	Indicator short mode

Table 8.28: Implementation Data Type - IndicatorStatusType

8.1.2.11 InitMonitorReasonType

Name	InitMonitorReasonType	
Kind	TYPE	
Derived from	uint8	
Description	InitMonitorReasonType	
Range / Symbol	*Limit*	*Description*
kClear	0x00	Event was cleared and all internal values and states are reset.
kRestart	0x01	Operation cycle of the event was (re-)started
kReenabled	0x02	Enable conditions or DTC settings re-enabled.
kStorageReenabled	0x03	Storage condition reenabled.

Table 8.29: Implementation Data Type - InitMonitorReasonType

8.1.2.12 OperationCycleStateType

Name	OperationCycleStateType	
Kind	TYPE	
Derived from	uint8	
Description	Represents the state information of operation cycles.	
Range / Symbol	*Limit*	*Description*
kStart	0x00	Start/restart the operation cycle.
kEnd	0x01	End the operation cycle

Table 8.30: Implementation Data Type - OperationCycleStateType

8.1.2.13 SnapshotDataRecordType

Name	SnapshotDataRecordType
Kind	STRUCTURE
Subelements	snapshotRecordNumber `uint8` snapshotDataElements `ByteVectorType`
Derived from	-
Description	Type containing a snapshot record number and its data of the snapshot record

Table 8.31: Implementation Data Type - SnapshotDataRecordType

8.1.2.14 SnapshotRecordUpdatedType

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

Table 8.32: Implementation Data Type - SnapshotRecordUpdatedType

8.1.3 Diagnostic Over IP

8.1.3.1 GIDstatusType

Name	GIDstatusType
Kind	STRUCTURE
Subelements	GID `GIDType` furtherActionReq `uint8` syncStatus `uint8`
Derived from	-
Description	Type used in the method confirmed for the status of the service processing

Table 8.33: Implementation Data Type - GIDstatusType

8.1.3.2 GIDType

Name	GIDType
Kind	ARRAY
Derived from	-
Description	

Table 8.34: Implementation Data Type - GIDType

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 DiagnosticServer

Port

Name	DiagnosticServer_{SoftwareCluster}		
Kind	ProvidedPort	***Interface***	DiagnosticServer
Description	Provides information about the ControlDTCStatus		
Variation	For each SoftwareCluster.		

Table 8.35: Port - DiagnosticServer_{SoftwareCluster}

Service Interface

Name	DiagnosticServer

Table 8.36: Service Interfaces - DiagnosticServer

Method

Name	EnableControlDTC
Description	Enforce restoring DTC setting.

Table 8.37: Service Interface DiagnosticServer - Method: EnableControlDTC

Fields

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

Table 8.38: Service Interface DiagnosticServer - Field: ControlDTCStatus

8.2.1.2 DiagnosticConversation

Port

Name	DiagnosticConversation_{SoftwareCluster}_{Number}		
Kind	ProvidedPort	***Interface***	DiagnosticConversation
Description	Provides information about diagnostic protocols.		
Variation	For each DiagnosticConversation within each SoftwareCluster.		

Table 8.39: Port - DiagnosticConversation_{SoftwareCluster}_{Number}

Service Interface

Name	DiagnosticConversation

Table 8.40: Service Interfaces - DiagnosticConversation

Method

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

Table 8.41: Service Interface DiagnosticConversation - Method: Cancel

Fields

Name	Identifier
Description	Contains the identifier (i.e. DiagnosticProtocolKind, TargetAddress, SourceAddress) of the diagnostic conversation.
Type	`DiagnosticConversationIdentifierType`
HasGetter	true

HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 8.42: Service Interface DiagnosticConversation - Field: Identifier

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

Table 8.43: Service Interface DiagnosticConversation - Field: Status

8.2.1.3 GenericUDSService

Port

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

Table 8.44: Port - GenericUDSService_{SoftwareCluster}_{DiagnosticServiceSwMapping}

Service Interface

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

Table 8.45: Service Interfaces - GenericUDSService

Method

Name	HandleMessage	
Description	Called for any request messsage of the mapped DiagnosticServiceSwMapping.	
Parameter	SID	
	Description	Diagnostic Request Service Identifier.
	Type	`SIDType`
	Variation	-
	Direction	IN
Parameter	requestData	
	Description	Diagnostic request data.
	Type	`ByteVectorType`
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	-
	Direction	IN
Parameter	responseData	
	Description	Diagnostic response data.
	Type	`ByteVectorType`
	Variation	-
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.46: Service Interface GenericUDSService - Method: HandleMessage

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

Table 8.47: Service Interface GenericUDSService - Method: Cancel

8.2.1.4 ServiceValidation

Port

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

Table 8.48: Port - ServiceValidation_{SoftwareCluster}_{DiagnosticServiceSwMapping}

Service Interface

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

Table 8.49: Service Interfaces - ServiceValidation

Method

Name	Validate	
Description	Called for any request message.	
Parameter	requestData	
	Description	Diagnostic request message including SID.
	Type	`ByteVectorType`
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	-
	Direction	IN
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.50: 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`
	Variation	-
	Direction	IN

Table 8.51: Service Interface ServiceValidation - Method: Confirmation

8.2.1.5 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 8.52: 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 8.53: 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`
	Variation	-
	Direction	IN
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	DataElement within response message.
	Type	
	Variation	For each DiagnosticDataIdentifier 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 8.54: Service Interface DataIdentifier - Method: Read

Name	Write	
Description	Called for WriteDataByIdentifer request for this DiagnosticDataIdentifier (if configured).	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	-
	Direction	IN

Parameter	dataRecord_{DiagnosticDataElement}	
	Description	DataElement within the request message.
	Type	
	Variation	For each DiagnosticDataIdentifier 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 8.55: Service Interface DataIdentifier - Method: Write

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

Table 8.56: Service Interface DataIdentifier - Method: Cancel

8.2.1.6 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 8.57: 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 8.58: 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	
	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`
	Variation	-
	Direction	IN
Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	
	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 8.59: 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	`uint8`
	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`
	Variation	-
	Direction	IN
Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the start sub-function according to DiagnosticRoutine.
	Type	
	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 8.60: Service Interface RoutineService - Method: Stop

Name	RequestResults	
Description	Called for sub-function requestRoutineResults of a routine if configured.	
Variation	Let Method Routine.getAttribut("requestResults").	
Parameter	req_{DiagnosticDataElement}	
	Description	IN-Parameter of the requestResults sub-function according to Diagnostic Routine.
	Type	
	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`
	Variation	-
	Direction	IN

Parameter	resp_{DiagnosticDataElement}	
	Description	OUT-Parameter of the requestRoutineResults sub-function according to DiagnosticRoutine.
	Type	
	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 8.61: Service Interface RoutineService - Method: RequestResults

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

Table 8.62: Service Interface RoutineService - Method: Cancel

8.2.1.7 SecurityAccess

Port

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

Table 8.63: Port - SecurityAccess_{SoftwareCluster}_{DiagnosticSecurityLevel}

Service Interface

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

Table 8.64: 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	`ByteVectorType`
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	-
	Direction	IN
Parameter	seed	
	Description	provided seed
	Type	`ByteVectorType`
	Variation	-
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.65: 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	`ByteVectorType`
	Variation	-
	Direction	IN
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	-
	Direction	IN
Parameter	result	
	Description	Result of the key validation.
	Type	`KeyCompareResultType`
	Variation	-
	Direction	OUT
Possible Application Errors	UDSService Failed	errorContext of UDSServiceFailed is of Type UDSResponseCodeType.

Table 8.66: Service Interface SecurityAccess - Method: CompareKey

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

Table 8.67: Service Interface SecurityAccess - Method: Cancel

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 8.68: Port - DiagnosticMonitor_{SoftwareCluster}_{DiagnosticEvent}

Service Interface

Name	DiagnosticMonitor

Table 8.69: 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 8.70: Service Interface DiagnosticMonitor - Method: InitMonitor

Event

Name	MonitorAction
Description	Contains either the last (un-)qulified test result of the diagnostic monitor or commands to control the debouncing or to force a prestorage.
Type	`MonitorActionType`

Table 8.71: 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 8.72: Port - DiagnosticEvent_{SoftwareCluster}_{DiagnosticEvent}

Service Interface

Name	DiagnosticEvent

Table 8.73: 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 8.74: 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	`FaultDetectionCounterType`
	Variation	-
	Direction	OUT

Table 8.75: Service Interface DiagnosticEvent - Method: FaultDetectionCounter

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

Table 8.76: 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
	Variation	-
	Direction	IN
Parameter	DTCOfEvent	
	Description	DTC number in respective DTCFormatType
	Type	DTCType
	Variation	-
	Direction	OUT

Table 8.77: 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 8.78: Port - DTCInformation_{SoftwareCluster}_{DiagnosticMemoryDestination}

Service Interface

Name	DTCInformation

Table 8.79: Service Interfaces - DTCInformation

Method

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

Table 8.80: Service Interface DTCInformation - Method: GetCurrentStatus

Events

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

Table 8.81: Service Interface DTCInformation - Event: DTCStatusChanged

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

Table 8.82: Service Interface DTCInformation - Event: SnapshotRecordUpdated

8.2.2.4 DiagnosticEventMemory

Port

Name	DiagnosticEventMemory_{SoftwareCluster}_{DiagnosticMemoryDestination}		
Kind	ProvidedPort	***Interface***	DiagnosticEventMemory
Description	Provides access to the fault memories.		
Variation	For each SoftwareCluster get all FaultMemories (DiagnosticMemoryDestinations) of associated DEXT.		

Table 8.83: Port - DiagnosticEventMemory_{SoftwareCluster}_{DiagnosticMemoryDestination}

Service Interface

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

Table 8.84: Service Interfaces - DiagnosticEventMemory

Field

Name	NumberOfStoredEventEntries
Description	Numbner of currently stored fault memory entries.
Type	`uint32`
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	0

Table 8.85: Service Interface DiagnosticEventMemory - Field: NumberOfStoredEvent Entries

Method

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

Table 8.86: Service Interface DiagnosticEventMemory - 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 8.87: Port - EnableCondition_{SoftwareCluster}_{DiagnosticEnableCondition}

Service Interface

Name	EnableCondition

Table 8.88: Service Interfaces - EnableCondition

Field

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

Table 8.89: 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 8.90: Port - StorageCondition_{SoftwareCluster}_{DiagnosticStorageCondition}

Service Interface

Name	StorageCondition

Table 8.91: Service Interfaces - StorageCondition

Field

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

Table 8.92: 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 8.93: Port - OperationCycle_{SoftwareCluster}_{DiagnosticOperationCycle}

Service Interface

Name	OperationCycle

Table 8.94: 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 8.95: 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 8.96: Port - Indicator_{SoftwareCluster}_{DiagnosticIndicator}

Service Interface

Name	Indicator

Table 8.97: 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 8.98: 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 8.99: Port - DataElement_{SoftwareCluster}_{DiagnosticDataElement}

Service Interface

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

Table 8.100: Service Interfaces - DataElement

Method

Name	Read	
Description	Called for data aquisition of a DiagnosticDataElement.	
Parameter	metaInfo	
	Description	MetaInfo of the request.
	Type	`MetaInfoType`
	Variation	-
	Direction	IN
Parameter	dataRecord_{DiagnosticDataElement}	
	Description	OUT-Parameter.
	Type	
	Variation	-
	Direction	OUT

Table 8.101: Service Interface DataElement - Method: Read

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

Table 8.102: Service Interface DataElement - Method: Cancel

8.2.3 DoIP protocol

8.2.3.1 DoIPGroupIdentification

Port

Name	DoIPGroupIdentification		
Kind	RequiredPort	***Interface***	DoIPGroupIdentification
Description	DoIPGroupIdentification		

Table 8.103: Port - DoIPGroupIdentification

Service Interface

Name	DoIPGroupIdentification

Table 8.104: Service Interfaces - DoIPGroupIdentification

Field

Name	GIDstatus
Description	Contains the current GID state for the DoIP protocol.
Type	`GIDstatusType`
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 8.105: Service Interface DoIPGroupIdentification - Field: GIDstatus

8.2.3.2 DoIPPowerModeInformation

Port

Name	DoIPPowerModeInformation		
Kind	RequiredPort	***Interface***	DoIPPowerModeInformation
Description	DoIPPowerModeInformation		

Table 8.106: Port - DoIPPowerModeInformation

Service Interface

Name	DoIPPowerModeInformation

Table 8.107: Service Interfaces - DoIPPowerModeInformation

Field

Name	PowerMode

Description	Contains the current power state for the DoIP protocol.
Type	`uint8`
HasGetter	true
HasNotifier	true
HasSetter	false
Init-Value	To be done: specify value

Table 8.108: Service Interface DoIPPowerModeInformation - Field: PowerMode

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.h` 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_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_00384] IndicationResult type ⌈ The UdsTransportProtocolMgr shall provide an public enumeration type for an UDS message indication result.

```
enum class IndicationResult : std::uint8_t {
  kIndicationOk = 0, kIndicationBusy = 1, kIndicationOverflow = 2
};
```

⌋()

[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 std::pair<IndicationResult, 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.h. ⌋()

[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_
transport_protocol_mgr);
```

⌋()

[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

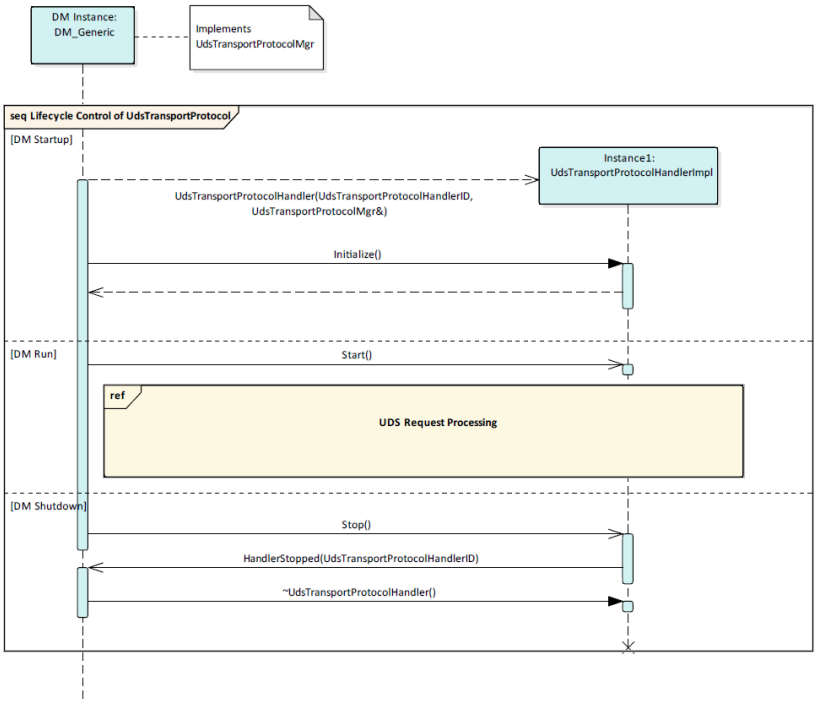

Figure 9.1: UDS Transport Life cycle

9.1.2 UDS Request Processing

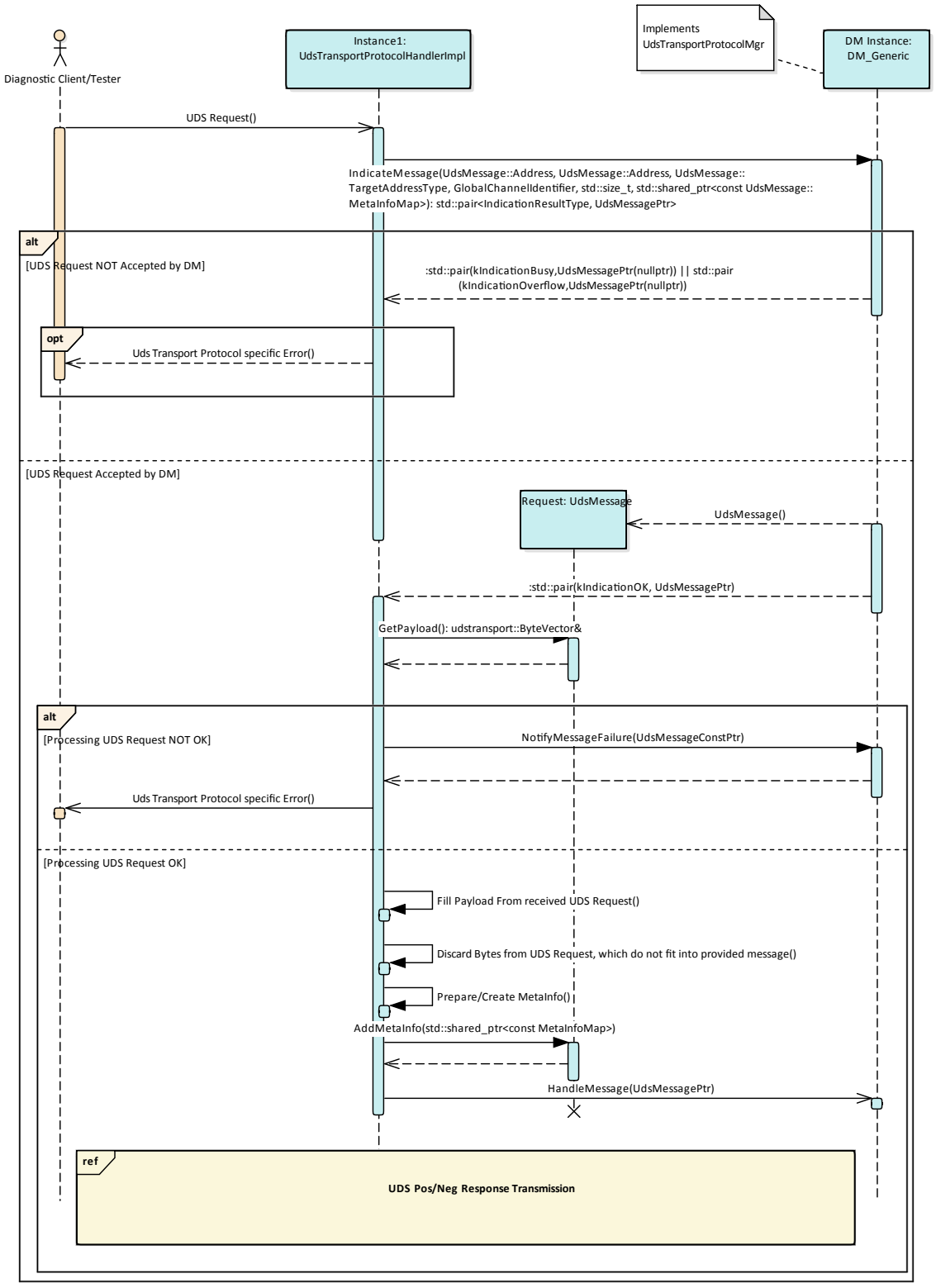

Figure 9.2: UDS Transport Request Processing

9.1.3 UDS Response Transmission

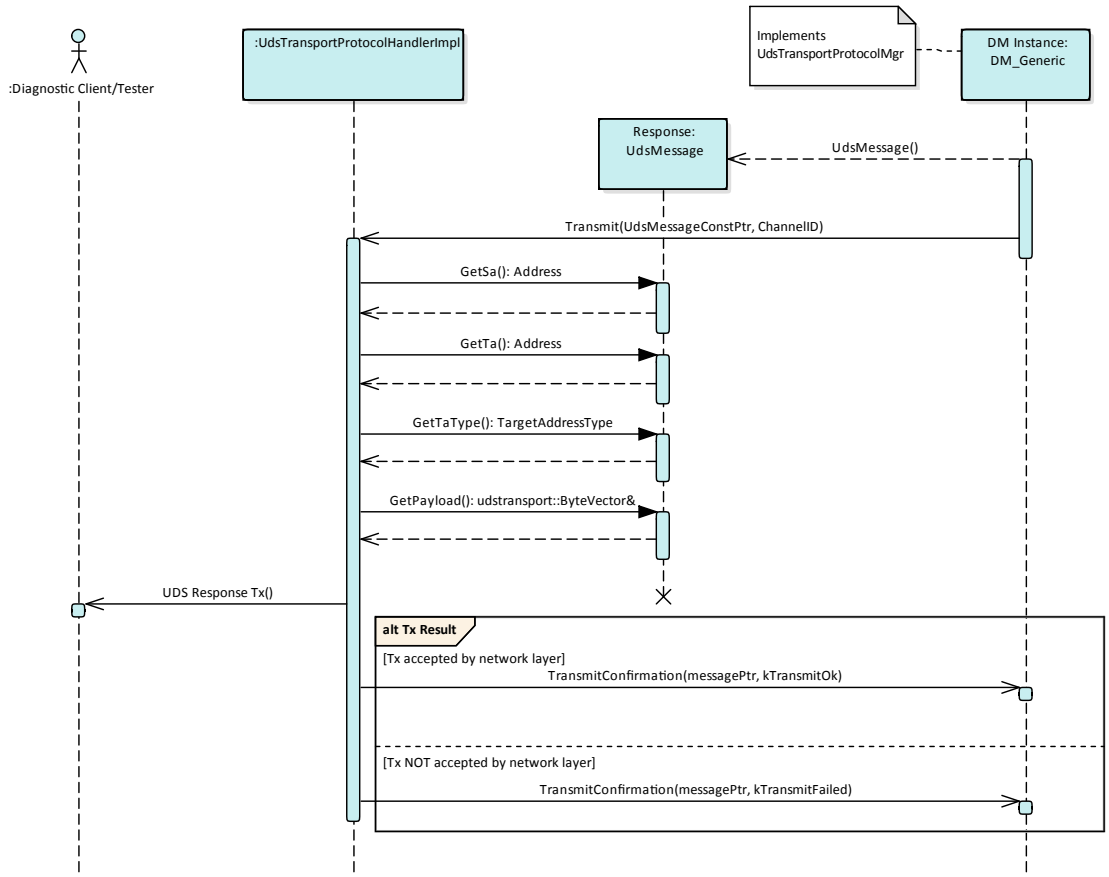

Figure 9.3: UDS Response Transmission

9.1.4 Channel Reestablishment

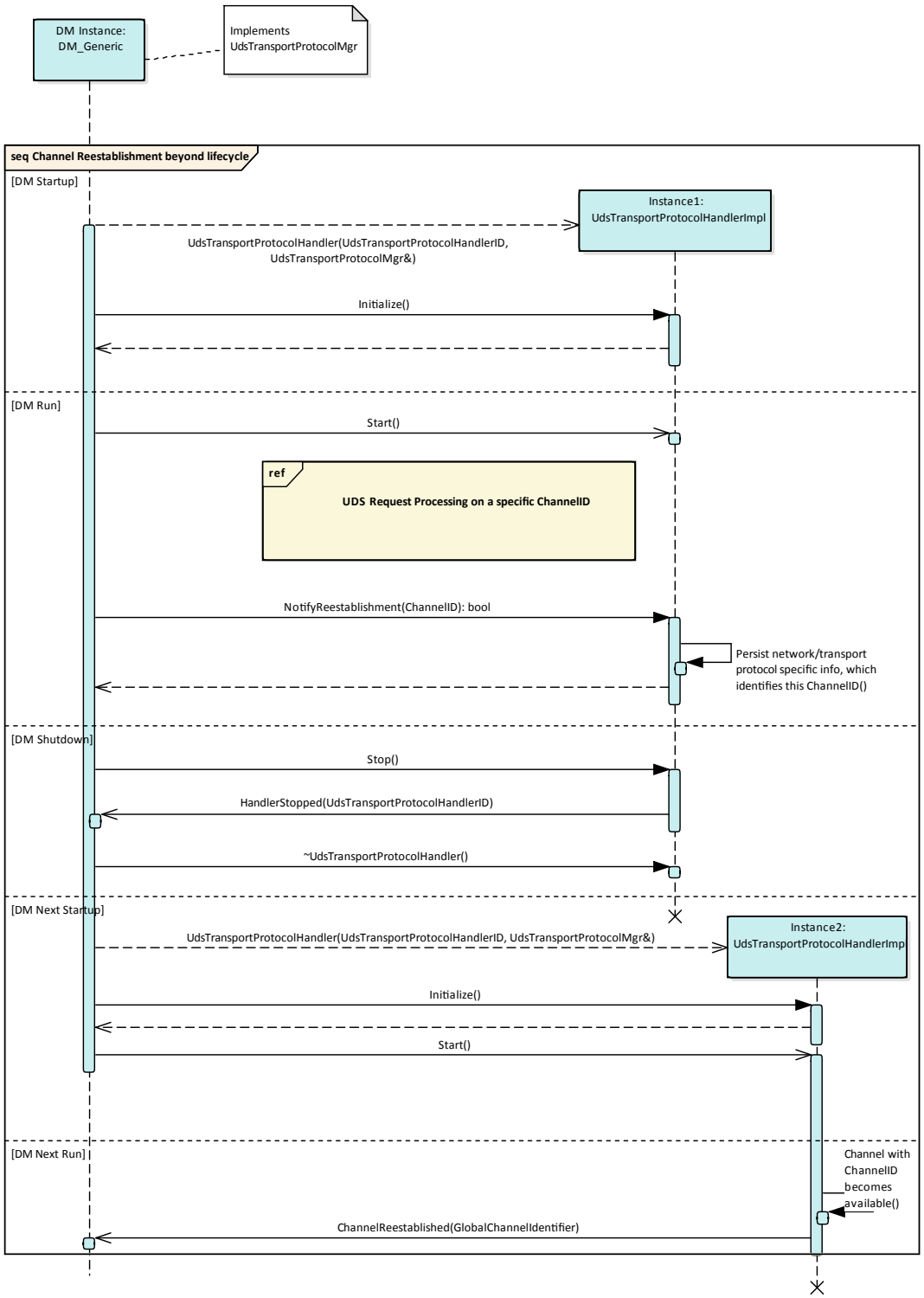

Figure 9.4: UDS Transport Channel Reestablishment

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	***DataPrototype* (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of any data type.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
swDataDefProps	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level.

Table A.1: DataPrototype

Class	**DiagEventDebounceCounterBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the counter-based debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemDebounceAlgorithmClass to 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.

Attribute	Type	Mul.	Kind	Note
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.2: DiagEventDebounceCounterBased

Class	DiagEventDebounceTimeBased			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the time-based pre-debounce algorithm shall be used by the Dem for this diagnostic monitor. This is related to set the EcuC choice container DemDebounceAlgorithmClass to 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.3: DiagEventDebounceTimeBased

Class	*DiagnosticAbstractDataIdentifier* (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class represents an abstract base class for the modeling of a diagnostic data identifier (DID).			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticDataIdentifier, DiagnosticDynamicDataIdentifier			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
id	PositiveInteger	1	attr	This is the numerical identifier used to identify the DiagnosticAbstractDataIdentifier in the scope of diagnostic workflow

Table A.4: DiagnosticAbstractDataIdentifier

Class	**DiagnosticAccessPermission**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm			
Note	This represents the specification of whether a given service can be accessed according to the existence of meta-classes referenced by a particular DiagnosticAccessPermission. In other words, this meta-class acts as a mapping element between several (otherwise unrelated) pieces of information that are put into context for the purpose of checking for access rights. **Tags:** atp.recommendedPackage=DiagnosticAccessPermissions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *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.5: DiagnosticAccessPermission

Class	**DiagnosticAging**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticAging			
Note	Defines the aging algorithm. **Tags:** atp.recommendedPackage=DiagnosticAgings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *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.6: DiagnosticAging

Enumeration	**DiagnosticClearDtcLimitationEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticCommonProps
Note	Scope of the DEM_ClearDTC Api.
Literal	***Description***

allSupported Dtcs	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.7: DiagnosticClearDtcLimitationEnum

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

Table A.8: DiagnosticClearEventBehaviorEnum

Class	**DiagnosticComControl**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::CommunicationControl			
Note	This represents an instance of the "Communication Control" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticCommunicationControls			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
comControl Class	DiagnosticCom ControlClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticComControl in the given context.
customSub FunctionNu mber	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.9: 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.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
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.10: 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.11: 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.12: DiagnosticControlDTCSetting

Class	***DiagnosticDataByIdentifier* (abstract)**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataByIdentifier			
Note	This represents an abstract base class for all diagnostic services that access data by identifier.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticReadDataByIdentifier, DiagnosticWriteDataByIdentifier			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataIdentifier	DiagnosticAbstractDataIdentifier	1	ref	This represents the linked DiagnosticDataIdentifier.

Table A.13: 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.14: 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***

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.15: DiagnosticDataIdentifier

Class	DiagnosticDataIdentifierSet			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This represents the ability to define a list of DiagnosticDataIdentifiers that can be reused in different contexts. **Tags:** atp.recommendedPackage=DiagnosticDataIdentifierSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataIdentifier (ordered)	DiagnosticDataIdentifier	*	ref	Reference to an orderd list of Data Identifiers.

Table A.16: DiagnosticDataIdentifierSet

Class	DiagnosticDebounceAlgorithmProps			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticDebouncingAlgorithm			
Note	Defines properties for the debounce algorithm class.			
Base	*ARObject*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
debounceAlgorithm	DiagEventDebounceAlgorithm	1	aggr	This represents the actual debounce algorithm.
debounceBehavior	DiagnosticDebounceBehaviorEnum	1	attr	This attribute defines how the event debounce algorithm will behave, if a related enable condition is not fulfilled or ControlDTCSetting of the related event is disabled.
debounceCounterStorage	Boolean	0..1	attr	Switch to store the debounce counter value non-volatile or not. true: debounce counter value shall be stored non-volatile false: debounce counter value is volatile

Table A.17: 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.18: DiagnosticDebounceBehaviorEnum

Class	**DiagnosticDemProvidedDataMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::ServiceMapping			
Note	This represents the ability to define the nature of a data access for a DiagnsoticDataElement in the Dem. **Tags:** atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataElement	DiagnosticDataElement	0..1	ref	This represents the DiagnosticDataElement for which the access is further qualified by the DiagnosticDemProvidedDataMapping.
dataProvider	NameToken	1	attr	This represents the ability to further specify the access within the Dem.

Table A.19: DiagnosticDemProvidedDataMapping

Class	**DiagnosticEcuReset**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EcuReset			
Note	This represents an instance of the "ECU Reset" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticEcuResets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
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.

Attribute	Type	Mul.	Kind	Note
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.20: 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.21: 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.22: DiagnosticEnableCondition

Class	***DiagnosticEnvCompareCondition* (abstract)**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EnvironmentalCondition			
Note	DiagnosticCompareConditions are atomic conditions. They are based on the idea of a comparison at runtime of some variable data with something constant. The type of the comparison (==, !=, <, <=, ...) is specified in DiagnosticCompareCondition.compareType.			
Base	*ARObject*, *DiagnosticEnvConditionFormulaPart*			
Subclasses	DiagnosticEnvDataCondition, DiagnosticEnvModeCondition			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
compareType	DiagnosticCompareTypeEnum	1	attr	This attributes represents the concrete type of the comparison.

Table A.23: 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.24: 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.25: 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.26: 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.

Attribute	Type	Mul.	Kind	Note
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.
prestorageFreezeFrame	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
recoverableInSameOperationCycle	Boolean	0..1	attr	If the attribute is set to true then reporting PASSED will reset the indication of a failed test in the current operation cycle. If the attribute is set to false then reporting PASSED will be ignored and not lead to a reset of the indication of a failed test.

Table A.27: 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
requiresCallbackExecution	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.28: 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.29: 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.30: 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.31: 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.32: 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.33: 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.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
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.34: 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.35: 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.36: DiagnosticIndicator

Class	***DiagnosticMapping* (abstract)**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticMapping			
Note	Abstract element for different kinds of diagnostic mappings.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticDemProvidedDataMapping, DiagnosticEventToDebounceAlgorithm Mapping, DiagnosticEventToEnableConditionGroupMapping, DiagnosticEventTo OperationCycleMapping, DiagnosticEventToStorageConditionGroupMapping, DiagnosticEventToTroubleCodeUdsMapping, DiagnosticFimAliasEventGroup Mapping, DiagnosticFimAliasEventMapping, DiagnosticInhibitSourceEventMapping, DiagnosticJ1939SpnMapping, DiagnosticServiceDataMapping, *DiagnosticSw Mapping*, DiagnosticTroubleCodeUdsToTroubleCodeObdMapping			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.37: DiagnosticMapping

Class	***DiagnosticMemoryDestination* (abstract)**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This abstract meta-class represents a possible memory destination for a diagnostic event.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticMemoryDestinationMirror, DiagnosticMemoryDestinationPrimary, DiagnosticMemoryDestinationUserDefined			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.38: 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*, *Diagnostic MemoryDestination*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.39: 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.40: 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.

Table A.41: 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

Attribute	Type	Mul.	Kind	Note
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.42: 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.
sendRespPendOnTransToBoot	Boolean	0..1	attr	The purpose of this attribute is to define whether or not the ECU should send a NRC 0x78 (response pending) before transitioning to the bootloader (in this case the attribute shall be set to "true") or if the transition shall be initiated without sending NRC 0x78 (in this case the attribute shall be set to "false").

Attribute	Type	Mul.	Kind	Note
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.43: 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.44: DiagnosticReadDTCInformation

Class	DiagnosticReadDataByIdentifier			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataByIdentifier			
Note	This represents an instance of the "Read Data by Identifier" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticDataByIdentifiers			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticDataByIdentifier*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
readClass	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.45: DiagnosticReadDataByIdentifier

Class	**DiagnosticReadDataByIdentifierClass**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataBy Identifier			
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.46: DiagnosticReadDataByIdentifierClass

Enumeration	**DiagnosticResponseToEcuResetEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EcuReset
Note	
Literal	***Description***
respondAfterReset	Answer to EcuReset service should come after the reset. **Tags:** atp.EnumerationValue=0
respondBeforeReset	Answer to EcuReset service should come before the reset. **Tags:** atp.EnumerationValue=1

Table A.47: DiagnosticResponseToEcuResetEnum

Class	**DiagnosticRoutine**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class represents the ability to define a diagnostic routine. **Tags:** atp.recommendedPackage=DiagnosticRoutines			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
id	PositiveInteger	1	attr	This is the numerical identifier used to identify the DiagnosticRoutine in the scope of diagnostic workflow **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
requestResult	DiagnosticRequestRoutineResults	0..1	aggr	This represents the ability to request the result of a running routine.

Attribute	Type	Mul.	Kind	Note
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.48: DiagnosticRoutine

Class	***DiagnosticRoutineSubfunction* (abstract)**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics			
Note	This meta-class acts as an abstract base class to routine subfunctions.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DiagnosticRequestRoutineResults, DiagnosticStartRoutine, DiagnosticStopRoutine			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
accessPermission	DiagnosticAccessPermission	0..1	ref	This reference represents the access permission of the owning routine subfunction.

Table A.49: DiagnosticRoutineSubfunction

Class	**DiagnosticSecurityAccess**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::SecurityAccess			
Note	This represents an instance of the "Security Access" diagnostic service. **Tags:** atp.recommendedPackage=DiagnosticSecurityAccesss			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
requestSeedId	PositiveInteger	1	attr	This would be 0x01, 0x03, 0x05, ... The sendKey id can be computed by adding 1 to the requestSeedId
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.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
securityLevel	DiagnosticSecurityLevel	1	ref	This reference identifies the applicable security level for the security access. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=securityLevel

Table A.50: DiagnosticSecurityAccess

Class	**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.51: 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*			
Subclasses	DiagnosticClearDiagnosticInformationClass, DiagnosticClearResetEmissionRelatedInfoClass, DiagnosticComControlClass, DiagnosticControlDTCSettingClass, DiagnosticCustomServiceClass, DiagnosticDataTransferClass, DiagnosticDynamicallyDefineDataIdentifierClass, DiagnosticEcuResetClass, DiagnosticIoControlClass, DiagnosticReadDTCInformationClass, DiagnosticReadDataByIdentifierClass, DiagnosticReadDataByPeriodicIDClass, DiagnosticReadMemoryByAddressClass, DiagnosticRequestControlOfOnBoardDeviceClass, DiagnosticRequestCurrentPowertrainDataClass, DiagnosticRequestDownloadClass, DiagnosticRequestEmissionRelatedDTCClass, DiagnosticRequestEmissionRelatedDTCPermanentStatusClass, DiagnosticRequestFileTransferClass, DiagnosticRequestOnBoardMonitoringTestResultsClass, DiagnosticRequestPowertrainFreezeFrameDataClass, DiagnosticRequestUploadClass, DiagnosticRequestVehicleInfoClass, DiagnosticResponseOnEventClass, DiagnosticRoutineControlClass, DiagnosticSecurityAccessClass, DiagnosticSessionControlClass, DiagnosticTransferExitClass, DiagnosticWriteDataByIdentifierClass, DiagnosticWriteMemoryByAddressClass			
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.52: 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 resp. event/notifier semantics in ServiceInterfaces on the adaptive platform. **Tags:** atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
diagnosticDataElement	DiagnosticDataElement	1	ref	This represents the applicable payload that corresponds to the referenced DataPrototype in the role mappedDataElement or (in case of a usage on the adaptive platform) mappedApDataElement.

Attribute	Type	Mul.	Kind	Note
mappedAp DataEleme nt	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
mappedDat aElement	DataPrototype	0..1	iref	This represents the dataElement in the application software that is accessed for diagnostic purpose. This role is applicable on the classic platform.
process	ProcessDesign	0..1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process; atp.Status=draft

Table A.53: DiagnosticServiceDataMapping

Class	*DiagnosticServiceInstance* (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::Common Service			
Note	This represents a concrete instance of a diagnostic service.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticClearDiagnosticInformation, DiagnosticClearResetEmissionRelatedInfo, DiagnosticComControl, DiagnosticControlDTCSetting, *DiagnosticDataByIdentifier*, DiagnosticDynamicallyDefineDataIdentifier, DiagnosticEcuReset, DiagnosticIO Control, *DiagnosticMemoryByAddress*, DiagnosticReadDTCInformation, Diagnostic ReadDataByPeriodicID, DiagnosticRequestControlOfOnBoardDevice, Diagnostic RequestCurrentPowertrainData, DiagnosticRequestEmissionRelatedDTC, Diagnostic RequestEmissionRelatedDTCPermanentStatus, DiagnosticRequestFileTransfer, DiagnosticRequestOnBoardMonitoringTestResults, DiagnosticRequestPowertrain FreezeFrameData, DiagnosticRequestVehicleInfo, DiagnosticResponseOnEvent, DiagnosticRoutineControl, DiagnosticSecurityAccess, DiagnosticSessionControl			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
accessPer mission	DiagnosticAcc essPermission	0..1	ref	This represents the collection of DiagnosticAccessPermissions that allow for the execution of the referencing DiagnosticServiceInstance..
serviceClas s	DiagnosticServ iceClass	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.54: 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 resp. method semantics of ServiceInterface on the adaptive platform. **Tags:** atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *DiagnosticSwMapping*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *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)
process	ProcessDesign	0..1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process; atp.Status=draft
serviceInstance	DiagnosticServiceInstance	0..1	ref	This represents the service instance that needs to be considered in this diagnostics service mapping.

Table A.55: 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.56: 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.57: 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.58: 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

Attribute	Type	Mul.	Kind	Note
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.59: 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.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
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.60: 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.61: DiagnosticWriteDataByIdentifier

Attribute	Type	Mul.	Kind	Note
Class	***Identifiable* (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractEvent*, *AbstractServiceInstance*, Action, *ActionItem*, ActionList, *AdaptiveModuleInstantiation*, AdaptiveSwcInternalBehavior, AliveSupervision, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, Arbitration, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *Atp Classifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildAction Environment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, Checkpoint Transition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, CppImplementationDataTypeElement, CryptoJob, CryptoKeySlot, CryptoNeedToCryptoJobMapping, CryptoPrimitive, Data PrototypeGroup, DataTransformation, DeadlineSupervision, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticData Element, DiagnosticFunctionInhibitSource, *DiagnosticRoutineSubfunction*, DoIpLogic Address, E2EProfileConfiguration, ECUMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, End2EndEvent ProtectionProps, EndToEndProtection, EventMapping, ExclusiveArea, *Executable Entity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMap Condition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FM FeatureSelection, FieldMapping, FireAndForgetMapping, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalSupervision, GlobalTimeGateway, *Global TimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, Hw AttributeLiteralDef, HwPin, HwPinGroup, IPv6ExtHeaderFilterList, ISignalToIPdu Mapping, ISignalTriggering, *IdentCaption*, ImplementationDataTypeElement, InterfaceMapping, InternalTriggeringPoint, J1939SharedAddressCluster, J1939Tp Node, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, Local Supervision, LogicalExpression, LogicalSupervision, MacMulticastGroup, McData Instance, MemorySection, MethodMapping, ModeDeclaration, ModeDeclaration Mapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, NvBlock Descriptor, *PackageableElement*, ParameterAccess, PduToFrameMapping, Pdu Triggering, PerInstanceMemory, PersistencyFileProxy, PersistencyKeyValuePair, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, PresharedKeyIdentity, ProcessToMachineMapping, Processor, ProcessorCore, Psk IdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, *RestAbstract Endpoint*, RestElementDef, RestResourceDef, RootSwComponentPrototype, Root SwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, Rpt ExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, Rule, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJob Requirement, *SecureComProps*, SecureCommunicationAuthenticationProps, *Secure CommunicationDeployment*, SecureCommunicationFreshnessProps, *ServerCall Point*, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInstanceToSignal Mapping, *ServiceInterfaceElementMapping*, ServiceInterfaceElementSecureCom Config, ServiceInterfaceMapping, *ServiceMethodDeployment*, *ServiceNeeds*, Signal BasedFieldToISignalTriggeringMapping, SocketAddress, SomeipEventGroup, Someip ProvidedEventGroup, *SpecElementReference*, *StackUsage*, StartupConfig, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, TcpOptionFilterList, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource,				

Attribute	*Type*	*Mul.*	*Kind*	*Note*
desc	MultiLanguage OverviewParagraph	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
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30

Attribute	Type	Mul.	Kind	Note
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.62: Identifiable

Class	PPortPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
providedInterface	PortInterface	1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table A.63: PPortPrototype

Class	RPortPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port requiring a certain port interface.			
Base	*ARObject*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**

Attribute	*Type*	*Mul.*	*Kind*	*Note*
requiredInterface	PortInterface	1	tref	The interface that this port requires, i.e. the port depends on another port providing the specified interface. **Stereotypes:** isOfType

Table A.64: RPortPrototype

Class	**ServiceInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.Status=draft; atp.recommendedPackage=ServiceInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
optionalElement	ServiceInterfaceSubElement	*	aggr	This aggregation represents the collectionof optional elements within the scope of the enclosing ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=optionalElement, variationPoint.shortLabel; atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	This represents the collection of ApplicationErrors defined in the context of the enclosing ServiceInterface. **Tags:** atp.Status=draft

Table A.65: ServiceInterface

Class	**SoftwareCluster**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareCluster			
Note	This meta-class represents the ability to define an uploadable software-package, i.e. the SoftwareCluster shall contain all software and configuration for a given purpose. **Tags:** atp.Status=draft; atp.recommendedPackage=SoftwareClusters			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
containedARElement	ARElement	*	ref	This reference represents the collection of model elements that cannot derive from UploadablePackageElement and that contribute to the completeness of the definition of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft
containedFibexElement	FibexElement	*	ref	This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster. **Tags:** atp.Status=draft
containedPackageElement	UploadablePackageElement	*	ref	This reference identifies model elements that are required to complete the manifest content. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft
containedProcess	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster. **Tags:** atp.Status=draft
dependsOn	SoftwareClusterDependency	*	aggr	This aggregation can be taken to identify a dependency for the enclosing SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dependsOn; atp.Status=draft
design	SoftwareClusterDesign	*	ref	This reference represents the identification of all SoftwareClusterDesigns applicable for the enclosing SoftwareCluster. **Stereotypes:** atpUriDef **Tags:** atp.Status=draft
diagnosticAddress	SoftwareClusterDiagnosticAddress	*	aggr	This aggregation represents the collection of diagnostic addresses that apply for the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=diagnosticAddress; atp.Status=draft

Attribute	Type	Mul.	Kind	Note
diagnosticExtract	DiagnosticContributionSet	0..1	ref	This reference represents the definition of the diagnostic extract applicable to the referencing SoftwareCluster **Tags:** atp.Status=draft
subSoftwareCluster	SoftwareCluster	*	ref	This reference is used to identify the sub-SoftwareClusters of an "umbrella" SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=subSoftwareCluster; atp.Status=draft
version	String	1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster. The format of the version as well as how to tell a lower from a higher version is not prescribed by the AUTOSAR standard.

Table A.66: SoftwareCluster

Class	*SoftwareClusterDiagnosticAddress* (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareCluster			
Note	This meta-class represents the ability to define a diagnostic address in an abstract form. Sub-classes are supposed to clarify how the diagnostic address shall be defined according to the applicable addressing scheme (DoIP vs. CAN TP vs. ...). **Tags:** atp.Status=draft			
Base	*ARObject*			
Subclasses	SoftwareClusterDoipDiagnosticAddress			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
addressSemantics	SoftwareClusterDiagnosticAddressSemanticsEnum	1	attr	This attribute clarifies whether the address value shall be interpreted as a physical or a functional address.

Table A.67: SoftwareClusterDiagnosticAddress

Enumeration	SoftwareClusterDiagnosticAddressSemanticsEnum
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareCluster
Note	This meta-class defines a list of semantics for the interpretation of diagnostic addresses in the context of a SoftwareCluster. **Tags:** atp.Status=draft
Literal	**Description**
functionalAddress	This address represents a functional address. **Tags:** atp.EnumerationValue=1
physicalAddress	This address represents a physical address. **Tags:** atp.EnumerationValue=0

Table A.68: SoftwareClusterDiagnosticAddressSemanticsEnum

Class	≪`atpVariation`≫ **SwDataDefProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the 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

Attribute	*Type*	*Mul.*	*Kind*	*Note*
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	AbstractImplementationDataType	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

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

Attribute	*Type*	*Mul.*	*Kind*	*Note*
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
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

Attribute	Type	Mul.	Kind	Note
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.69: SwDataDefProps

Class	SwcServiceDependency			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	Specialization of ServiceDependency in the context of an SwcInternalBehavior. It allows to associate ports, port groups and (in special cases) data defined for an atomic software component to a given ServiceNeeds element.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceDependency*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
assignedData	RoleBasedDataAssignment	*	aggr	Defines the role of an associated data object of the same component. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
assignedPort	RoleBasedPortAssignment	*	aggr	Defines the role of an associated port of the same component. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedPort, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Attribute	*Type*	*Mul.*	*Kind*	*Note*
represented PortGroup	PortGroup	0..1	ref	This reference specifies an association between the ServiceNeeeds and a PortGroup, for example to request a communication mode which applies for communication via these ports. The referred PortGroup shall be local to this atomic SWC, but via the links between the PortGroups, a tool can evaluate this information such that all the ports linked via this port group on the same ECU can be found.
serviceNeeds	ServiceNeeds	1	aggr	The associated ServiceNeeds.

Table A.70: SwcServiceDependency