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

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R19-11

Document Change History

Date	Release	Changed by	Description
2019-11-28	R19-11	AUTOSAR Release Management	• Document quality improvement and fixing bugs • Incorporated Quality Scope Review Findings • Partly removed obsolete requirements • Removed obsolete service interfaces • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Document quality improvement and fixing bugs • Introduced ara::diag interfaces in draft state
2018-10-31	18-10	AUTOSAR Release Management	• Diagnostic Protocol replaced by Diagnostic Conversations • ResponseOnEvent, CommunicationControl, EcuReset added • Chapter 7 overall rework and updates • Chapter 8 split into chapter 8 (C++ API) and chapter 9 (Service Interfaces)

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

Disclaimer

Table of Contents

1 Introduction and functional overview

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

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

1.1 Diagnostic interface

Since release R19-03 a C++ interface was introduced for diagnostics as a replacement for the former ara::com based service interface.

1.2 AUTOSAR Diagnostic Extract Template (DEXT)

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

1.3 Software Cluster

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

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

1.3.1 Diagnostic Server

The Diagnostic Communication 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 Communication Management function block of DM
  - or process the service internally within Diagnostic Event 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 Communication Management part.

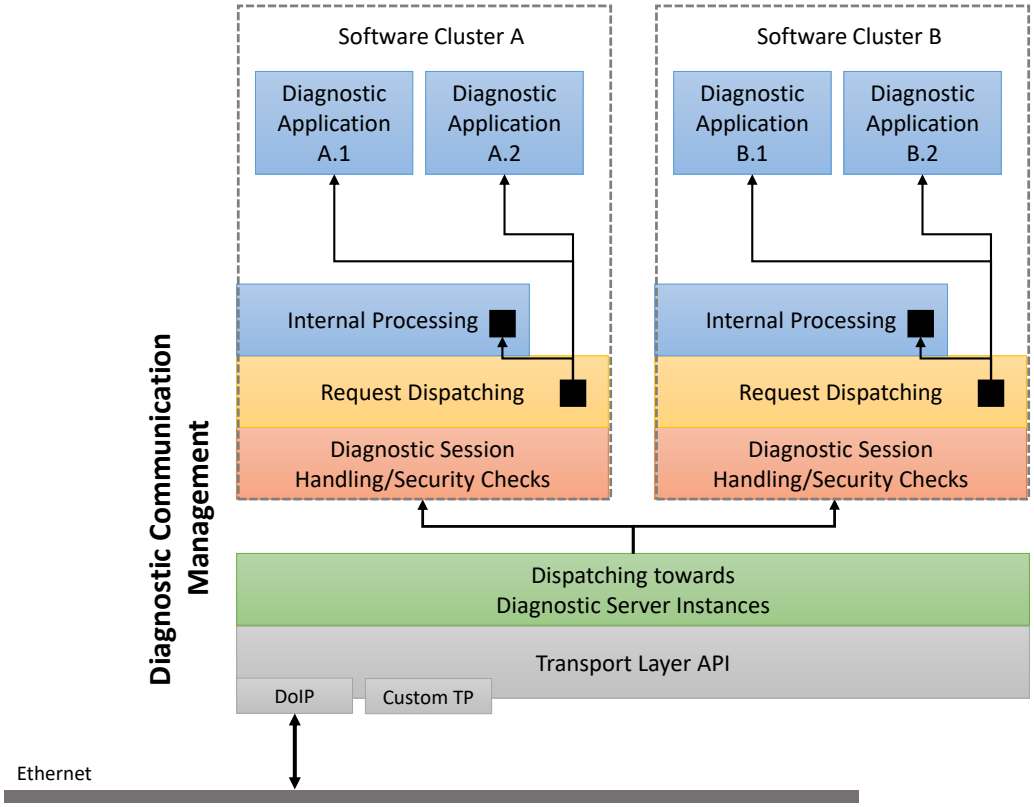

Figure 1.1: Architecture Diagnostic Communication Management

1.3.2 Diagnostic Managers external dependencies

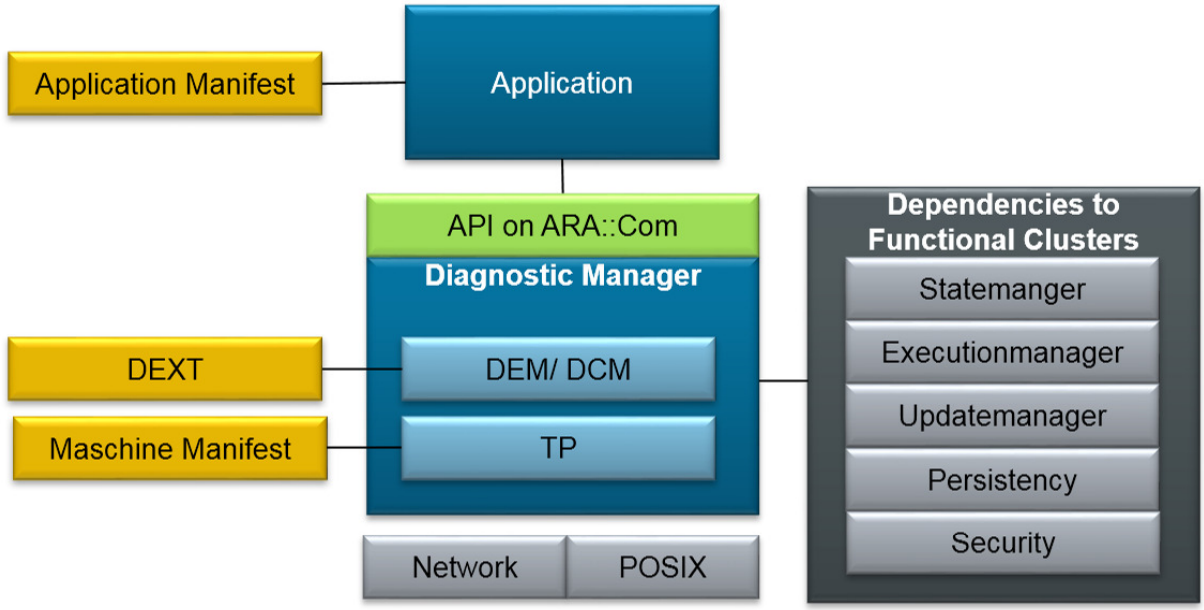

Figure 1.2: Diagnostic Managers external dependencies

2 Acronyms and Abbreviations

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

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

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

Terms:	Description:
Aging	Unlearning/deleting of a no longer failed event/DTC after a defined number of operation cycles from event memory.
Associated ServiceInterface	Describes the association of a ServiceInterface to a DiagnosticServiceSwMapping by means of a referenced SwcServiceDependency, see section 7.2.4.2.1.
Diagnostic Client	A Diagnostic Client is a diagnostic service requester, i.e. sends a UDS request to the Diagnostic Server. Usually the Diagnostic Client is an external tester equipment but can also be another vehicle internal ECU.
Diagnostic Communication Management	Diagnostic Communication Management is the part of the Diagnostic Management which belongs to tester communication and the processing of UDS services.
Diagnostic Conversation	Diagnostic Conversation represents a conversation between Diagnostic Client (Tester) and Diagnostic Server.
Diagnostic Event Management	Diagnostic Event Management is the part of the Diagnostic Management which belongs to processing and storing of diagnostic events and associated data.
Diagnostic Management	Diagnostic Management is a placeholder for the complete functionality of diagnostic communication and event handling.
Diagnostic Server instance	Diagnostic Server (DM) is intended to support an own Diagnostic Server instance per installed *SoftwareCluster*, see section 7.2 for a detailed description. Each of those Server instances has and manages its own resources and is responsible for dispatching and processing of diagnostic services.
Diagnostic Service instance	A diagnostic service instance implements a concrete use of a diagnostic service in a given context. It refers to a DiagnosticServiceClass and the DiagnosticAccessPermission , see 7.2.1.3.3 for a detailed description.

Terms:	Description:
DTC group	Uniquely identifies a set of DTCs. A DTC group is mapped to the range of valid DTCs. By providing a group of DTCs it is expressed that a certain operation is requested on all DTCs of that group. The DTC group definition is provided by ISO 14229-1[1] and OEM/supplier-specific.
Enable Conditions	The criteria / conditions under which the test results from the monitors in the AA's are valid and shall be processed by DM. Configuration is done per event.
Extended Data Records	Contains statistical data for a DTC. Extended data records are assigned to DTCs and maintained and stored by the DM.
Event	An *event* (also *diagnostic event*) uniquely identifies a fault path of the system. An application monitors the system and reports events to the DM.
Event memory	The DM stores information about events in the event memory. There can be multiple event memories, each keeping information independently from each other. Examples of the event memory is the UDS primary event memory or the up to 256 user-defined event memories.
GroupOfAllDTCs	Identifies a special DTC group that contains all DTCs. This DTC group is identified by the DTC value 0xFFFFFF in 14229-1[1] and contains by default all DTCs of a fault memory. It is present by default in the DM and requires no configuration.
Internal, External	Classifies if a DiagnosticDataElement is either managed internally inside DM or by an external adaptive applications, see 7.2.4.1 for the precise definition.
Internally, Externally	Definition of the support type of a SID by the DM. Internally means processing is done by DM itself, Externally means an external service processor is used.
Monitor	A *monitor* (also *diagnostic 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.
SourceAddress	A Source Address is used to encode client and server identifiers. In a UDS request the source address encodes the Diagnostic Client whereas the source address in a UDS response encodes the Diagnostic Server.

Terms:	Description:
TargetAddress	A Target Address is used to encode client and server identifiers. In a UDS request the target address encodes the Diagnostic Server whereas the target address in a UDS response encodes the Diagnostic Client.
Transport Protocol Handler	A subcomponent of DM implementing a particular Transport Protocol (either DoIP or any other proprietary UDS Transport Layer).
Transport Protocol Manager	Link between UDS Transport Layer and Application Layer.
UDS service	A diagnostic service as defined in ISO 14229-1[1].
UDS DTC status bit	UDS DTC status bit as defined in ISO 14229-1[1] Annex D.2; Each single bit position represents and documents a certain status information for the connected diagnostic event or DTC. The following eight bits are defined: **Nr:** Definition: **0** testFailed **1** testFailedThisOperationCycle **2** pendingDTC **3** confirmedDTC **4** testNotCompletedSinceLastClear **5** testFailedSinceLastClear **6** testNotCompletedThisOperationCycle **7** warningIndicatorRequested All eight bits constitute the UDS DTC status byte.
UDS DTC status byte	Bit-packed DTC status information byte as defined in ISO 14229-1[1], based on DTC level. Contains the UDS DTC status bits.
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.
Non-volatile Memory	In the context of DM, Non-volatile Memory refers to the persistent information over the shutdown of the DM process. This does not depend on HW details.

3 Related documentation

3.1 Input documents & related standards and norms

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

[2] Diagnostic Extract Template
AUTOSAR_TPS_DiagnosticExtractTemplate

[3] Glossary
AUTOSAR_TR_Glossary

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

[5] General Specification of Adaptive Platform
AUTOSAR_SWS_General

[6] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement

[7] Specification of Log and Trace
AUTOSAR_SWS_LogAndTrace

[8] Specification of Persistency
AUTOSAR_SWS_Persistency

[9] Requirements on Diagnostics
AUTOSAR_RS_Diagnostics

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

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

[12] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

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

[14] Specification of Core Types for Adaptive Platform
AUTOSAR_SWS_CoreTypes

3.2 Further applicable specification

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

4 Constraints and assumptions

4.1 Known Limitations

This chapter describes known limitation of the DM in respect to general claimed goals of the module. The nature of constraints can be a general exclusion of a certain 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 UDS services are not implemented by the DM:
  - 0x23 ReadMemoryByAddress
  - 0x24 ReadScalingDataByIdentifier
  - 0x2A ReadDataByPeriodicIdentifier
  - 0x2C DynamicallyDefineDataIdentifier
  - 0x2F InputOutputControlByIdentifier
  - 0x38 RequestFileTransfer
  - 0x3D WriteMemoryByAddress
  - 0x83 AccessTimingParameter
  - 0x84 SecuredDataTransmission
  - 0x87 LinkControl
- Sub-functions of UDS services are implemented according to ISO 14229-1[1] unless explicitly stated.
- The UDS mirror event memory is not supported by the DM. As a result of this, the DM does not support the UDS service.
  - 0x19 with subfunction 0x0F (reportMirrorMemoryDTCByStatusMask)
  - 0x19 with subfunction 0x10 (reportMirrorMemoryDTCExtDataRecordByDTCNumber)
  - 0x19 with subfunction 0x11 (reportNumberOfMirrorMemoryDTCByStatusMask)
- The OBD/WWH OBD is not supported by the DM. As a result of this, the DM does not support the UDS service.

  - 0x19 with subfunction 0x05 (reportDTCStoredDataByRecordNumber)
  - 0x19 with subfunction 0x12 (reportNumberOfEmissionsOBDDTCByStatus-Mask)
  - 0x19 with subfunction 0x13 (reportEmissionsOBDDTCByStatusMask)
  - 0x19 with subfunction 0x42 (reportWWHOBDDTCByMaskRecord)
  - 0x19 with subfunction 0x55 (reportWWHOBDDTCWithPermanentStatus)
- The following general UDS services are currently not supported, but still under discussion:
  - 0x19 with subfunction 0x03 (reportDTCSnapshotIdentification)
  - 0x19 with subfunction 0x08 (reportDTCBySeverityMaskRecord)
  - 0x19 with subfunction 0x09 (reportSeverityInformationOfDTC)
  - 0x19 with subfunction 0x0A (reportSupportedDTC)
  - 0x19 with subfunction 0x0B (reportFirstTestFailedDTC)
  - 0x19 with subfunction 0x0C (reportFirstConfirmedDTC)
  - 0x19 with subfunction 0x0D (reportMostRecentTestFailedDTC)
  - 0x19 with subfunction 0x0E (reportMostRecentConfirmedDTC)
  - 0x19 with subfunction 0x15 (reportDTCWithPermanentStatus)
  - 0x19 with subfunction 0x16 (reportDTCExtDataRecordByRecordNumber)
- Event Memory: Variant handling at runtime for events/DTCs is not supported.
- Event Memory: Details for combined events are not specified.
- Event Memory: Event displacement is not supported. The DM stores for each DTC related data.
- Event Memory: Internal configuration parameters and DM values as extended data are not supported.
- Persistent Storage of failed attempts to change security level : After each increment of the attempt counter, it shall be persisted to survive accidental or intended resets. Here the option to select the persistent storage is mandatory in Adaptive Autosar.

5 Dependencies to other modules

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

The DM may use ara::log ([7], Log and Trace) for logging and tracing purposes.

DM may use ara::per ([8], Persistency) to store non-volatile data.

6 Requirements Tracing

The following tables reference the requirements specified in [9] 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
[RS_AP_00125]	Enumerator and constant names.	[SWS_DM_00642] [SWS_DM_00643] [SWS_DM_00645]
[RS_AP_00134]	noexcept behavior of class destructors	[SWS_DM_00553] [SWS_DM_00584] [SWS_DM_00586] [SWS_DM_00588] [SWS_DM_00590] [SWS_DM_00635] [SWS_DM_00648] [SWS_DM_00665] [SWS_DM_00713] [SWS_DM_00723] [SWS_DM_00733] [SWS_DM_00743] [SWS_DM_00753] [SWS_DM_00763] [SWS_DM_00773] [SWS_DM_00788] [SWS_DM_00798] [SWS_DM_00807] [SWS_DM_00832]
[RS_AP_00137]	Connecting run-time interface with model.	[SWS_DM_00548] [SWS_DM_00549] [SWS_DM_00550] [SWS_DM_00552] [SWS_DM_00585] [SWS_DM_00587] [SWS_DM_00589] [SWS_DM_00616] [SWS_DM_00634] [SWS_DM_00647] [SWS_DM_00664] [SWS_DM_00712] [SWS_DM_00722] [SWS_DM_00732] [SWS_DM_00742] [SWS_DM_00752] [SWS_DM_00762] [SWS_DM_00772] [SWS_DM_00787] [SWS_DM_00797] [SWS_DM_00806]
[RS_AP_00138]	Return type of asynchronous function calls.	[SWS_DM_00554] [SWS_DM_00555] [SWS_DM_00557] [SWS_DM_00591] [SWS_DM_00592] [SWS_DM_00593] [SWS_DM_00596] [SWS_DM_00598] [SWS_DM_00618] [SWS_DM_00636] [SWS_DM_00637] [SWS_DM_00640] [SWS_DM_00724] [SWS_DM_00734] [SWS_DM_00764] [SWS_DM_00765] [SWS_DM_00774] [SWS_DM_00775] [SWS_DM_00789] [SWS_DM_00790] [SWS_DM_00791] [SWS_DM_00799] [SWS_DM_00800] [SWS_DM_00801] [SWS_DM_00808]

Requirement	Description	Satisfied by
[RS_AP_00139]	Return type of synchronous function calls.	[SWS_DM_00543] [SWS_DM_00594] [SWS_DM_00597] [SWS_DM_00599] [SWS_DM_00619] [SWS_DM_00638] [SWS_DM_00649] [SWS_DM_00650] [SWS_DM_00651] [SWS_DM_00652] [SWS_DM_00653] [SWS_DM_00654] [SWS_DM_00655] [SWS_DM_00656] [SWS_DM_00666] [SWS_DM_00667] [SWS_DM_00668] [SWS_DM_00669] [SWS_DM_00670] [SWS_DM_00671] [SWS_DM_00672] [SWS_DM_00673] [SWS_DM_00674] [SWS_DM_00692] [SWS_DM_00694] [SWS_DM_00695] [SWS_DM_00696] [SWS_DM_00697] [SWS_DM_00698] [SWS_DM_00699] [SWS_DM_00700] [SWS_DM_00714] [SWS_DM_00715] [SWS_DM_00725] [SWS_DM_00735] [SWS_DM_00744] [SWS_DM_00745] [SWS_DM_00754] [SWS_DM_00755] [SWS_DM_00756] [SWS_DM_00766] [SWS_DM_00776] [SWS_DM_00792] [SWS_DM_00802] [SWS_DM_00809]
[RS_Diag_04005]	Manage Security Access level handling	[SWS_DM_00047] [SWS_DM_00103] [SWS_DM_00236] [SWS_DM_00421] [SWS_DM_00760] [SWS_DM_00761] [SWS_DM_00762] [SWS_DM_00763] [SWS_DM_00764] [SWS_DM_00765] [SWS_DM_00766] [SWS_DM_00767]
[RS_Diag_04006]	Manage session handling	[SWS_DM_00046] [SWS_DM_00101] [SWS_DM_00102] [SWS_DM_00380] [SWS_DM_00381] [SWS_DM_00382] [SWS_DM_00383] [SWS_DM_00842]
[RS_Diag_04016]	Support ”Busy handling” by sending a negative response 0x78	[SWS_DM_00368] [SWS_DM_00369]
[RS_Diag_04019]	Provide confirmation after transmit diagnostic responses to the application	[SWS_DM_00268] [SWS_DM_00859]
[RS_Diag_04020]	Suppress responses to diagnostic tool requests	[SWS_DM_00365] [SWS_DM_00433] [SWS_DM_00862]
[RS_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_00868] [SWS_DM_00869] [SWS_DM_00870] [SWS_DM_00871] [SWS_DM_00872]
[RS_Diag_04059]	Configuration of timing parameters	[SWS_DM_NA]
[RS_Diag_04063]	Process a dedicated event identifier for each monitoring path to support an autonomous handling of different events/ faults	[SWS_DM_00007]

Requirement	Description	Satisfied by
[RS_Diag_04064]	Provide configurable buffer sizes for storage of the events, status information and environmental data	[SWS_DM_NA]
[RS_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_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374] [SWS_DM_00658] [SWS_DM_00659]
[RS_Diag_04068]	The diagnostic in AUTOSAR shall support event specific debounce counters to improve signal quality internally (According to ISO 14229-1 Appendix D)	[SWS_DM_00013] [SWS_DM_00014] [SWS_DM_00021] [SWS_DM_00022] [SWS_DM_00023] [SWS_DM_00024] [SWS_DM_00025] [SWS_DM_00029] [SWS_DM_00039] [SWS_DM_00040] [SWS_DM_00086] [SWS_DM_00538] [SWS_DM_00549] [SWS_DM_00645] [SWS_DM_00654] [SWS_DM_00656] [SWS_DM_00874] [SWS_DM_00875] [SWS_DM_00876] [SWS_DM_00879] [SWS_DM_00880]
[RS_Diag_04097]	Decentralized and modular diagnostic configuration in applications	[SWS_DM_00393] [SWS_DM_00401] [SWS_DM_00570] [SWS_DM_00572] [SWS_DM_00848] [SWS_DM_00849] [SWS_DM_00850] [SWS_DM_00903] [SWS_DM_00904] [SWS_DM_00905] [SWS_DM_00906] [SWS_DM_00907] [SWS_DM_00908] [SWS_DM_CONSTR_00394] [SWS_DM_CONSTR_00395] [SWS_DM_CONSTR_00396]
[RS_Diag_04105]	Event memory management	[SWS_DM_00148] [SWS_DM_00150] [SWS_DM_00657] [SWS_DM_00664]
[RS_Diag_04109]	Provide an interface to retrieve the number of event memory entries	[SWS_DM_00669] [SWS_DM_00670] [SWS_DM_00902]
[RS_Diag_04115]	The optional parameter DTCSettingControlOption Record as part of UDS service ControlDTCSetting shall be limited to GroupOfDTC	[SWS_DM_00064] [SWS_DM_00231]
[RS_Diag_04117]	Configurable behavior for DTC deletion	[SWS_DM_00064] [SWS_DM_00065] [SWS_DM_00091] [SWS_DM_00092] [SWS_DM_00116] [SWS_DM_00117] [SWS_DM_00121] [SWS_DM_00122] [SWS_DM_00123] [SWS_DM_00124] [SWS_DM_00144] [SWS_DM_00145] [SWS_DM_00146] [SWS_DM_00147] [SWS_DM_00159] [SWS_DM_00160] [SWS_DM_00896] [SWS_DM_CONSTR_00082]

Requirement	Description	Satisfied by
[RS_Diag_04119]	Handle the execution of diagnostic services according to the assigned diagnostic session	[SWS_DM_00046]
[RS_Diag_04120]	Support a predefined Address AndLengthFormatIdentifier	[SWS_DM_00129] [SWS_DM_00130]
[RS_Diag_04124]	Store the current debounce counter value non-volatile to over a power-down cycle	[SWS_DM_00018] [SWS_DM_00028]
[RS_Diag_04125]	Event debounce counter shall be configurable	[SWS_DM_00017] [SWS_DM_00021] [SWS_DM_00024] [SWS_DM_00025] [SWS_DM_00029] [SWS_DM_00088] [SWS_DM_00378] [SWS_DM_00564] [SWS_DM_00565] [SWS_DM_00875] [SWS_DM_00876] [SWS_DM_00881] [SWS_DM_00882]
[RS_Diag_04127]	Configurable record numbers and trigger options for DTCSnapshotRecords and DTCExtendedDataRecords	[SWS_DM_00893] [SWS_DM_00895]
[RS_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]
[RS_Diag_04136]	Configurable ”confirmed” threshold	[SWS_DM_00218]
[RS_Diag_04140]	Aging for UDS status bits ”confirmedDTC” and ”testFailed SinceLastClear”	[SWS_DM_00243]
[RS_Diag_04148]	Provide capabilities to inform applications about diagnostic data changes	[SWS_DM_00667] [SWS_DM_00894]
[RS_Diag_04150]	Support the primary fault memory defined by ISO 14229-1	[SWS_DM_00055] [SWS_DM_00056] [SWS_DM_00083] [SWS_DM_00657] [SWS_DM_00664] [SWS_DM_00911] [SWS_DM_CONSTR_00084]
[RS_Diag_04151]	Event status handling	[SWS_DM_00213] [SWS_DM_00643] [SWS_DM_00644] [SWS_DM_00646] [SWS_DM_00647] [SWS_DM_00648] [SWS_DM_00649] [SWS_DM_00652] [SWS_DM_00655] [SWS_DM_00658] [SWS_DM_00659] [SWS_DM_00883] [SWS_DM_00884]
[RS_Diag_04157]	Reporting of DTCs and related data	[SWS_DM_00058] [SWS_DM_00061] [SWS_DM_00062] [SWS_DM_00063] [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]
[RS_Diag_04159]	Control of DTC storage	[SWS_DM_00088] [SWS_DM_00229] [SWS_DM_00378] [SWS_DM_00565] [SWS_DM_00663] [SWS_DM_00672] [SWS_DM_00673] [SWS_DM_00674] [SWS_DM_00909] [SWS_DM_00910]

Requirement	Description	Satisfied by
[RS_Diag_04160]	ResponseOnEvent according to ISO 14229-1	[SWS_DM_00491] [SWS_DM_00492] [SWS_DM_00493] [SWS_DM_00494] [SWS_DM_00495] [SWS_DM_00496] [SWS_DM_00497] [SWS_DM_00498] [SWS_DM_00499] [SWS_DM_00500] [SWS_DM_00501]
[RS_Diag_04164]	Independent event memories for multiple diagnostic server instances (virtual ECUs)	[SWS_DM_00657] [SWS_DM_00664]
[RS_Diag_04166]	Several tester conversations in parallel with assigned priorities	[SWS_DM_00425] [SWS_DM_00426] [SWS_DM_00427] [SWS_DM_00428] [SWS_DM_00429] [SWS_DM_00430] [SWS_DM_00840] [SWS_DM_00841] [SWS_DM_00843] [SWS_DM_00844] [SWS_DM_00856]
[RS_Diag_04167]	Conversation preemption/ abortion	[SWS_DM_00049] [SWS_DM_00277] [SWS_DM_00278] [SWS_DM_00279] [SWS_DM_00280] [SWS_DM_00290] [SWS_DM_00431] [SWS_DM_00482] [SWS_DM_00577] [SWS_DM_00847]
[RS_Diag_04168]	Adding of user-defined transport layers	[SWS_DM_00329] [SWS_DM_00330] [SWS_DM_00331] [SWS_DM_00332] [SWS_DM_00333] [SWS_DM_00340] [SWS_DM_00342] [SWS_DM_00345] [SWS_DM_00346] [SWS_DM_00347] [SWS_DM_00348] [SWS_DM_00349] [SWS_DM_00350] [SWS_DM_00351] [SWS_DM_00356] [SWS_DM_00357] [SWS_DM_00358] [SWS_DM_00359] [SWS_DM_00385] [SWS_DM_00386] [SWS_DM_00387] [SWS_DM_00388] [SWS_DM_00389] [SWS_DM_00392] [SWS_DM_00487]
[RS_Diag_04169]	Provide an interface for external UDS service processors.	[SWS_DM_00865]
[RS_Diag_04170]	Provide connection specific meta information to external service processors	[SWS_DM_00294] [SWS_DM_00302] [SWS_DM_00554] [SWS_DM_00555] [SWS_DM_00556] [SWS_DM_00591] [SWS_DM_00592] [SWS_DM_00593] [SWS_DM_00596] [SWS_DM_00598] [SWS_DM_00618] [SWS_DM_00636] [SWS_DM_00637] [SWS_DM_00640] [SWS_DM_00692] [SWS_DM_00764] [SWS_DM_00765] [SWS_DM_00774] [SWS_DM_00775] [SWS_DM_00789] [SWS_DM_00790] [SWS_DM_00791] [SWS_DM_00799] [SWS_DM_00800] [SWS_DM_00801] [SWS_DM_00808]
[RS_Diag_04171]	Synchronous and asynchronous interaction with external service processors	[SWS_DM_NA]
[RS_Diag_04172]	Inform external service processors about outcome of the final response	[SWS_DM_00859]

Requirement	Description	Satisfied by
[RS_Diag_04177]	Custom diagnostic services	[SWS_DM_00502]
[RS_Diag_04178]	Support operation cycles according to ISO 14229-1	[SWS_DM_00004] [SWS_DM_00567] [SWS_DM_00750] [SWS_DM_00751] [SWS_DM_00752] [SWS_DM_00753] [SWS_DM_00754] [SWS_DM_00755] [SWS_DM_00756] [SWS_DM_00885] [SWS_DM_00889] [SWS_DM_00890] [SWS_DM_00891] [SWS_DM_00892] [SWS_DM_CONSTR_00168]
[RS_Diag_04179]	Provide interfaces for monitoring application.	[SWS_DM_00007] [SWS_DM_00540] [SWS_DM_00541] [SWS_DM_00542] [SWS_DM_00543] [SWS_DM_00548] [SWS_DM_00549] [SWS_DM_00550] [SWS_DM_00873]
[RS_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_00115] [SWS_DM_00162] [SWS_DM_00163] [SWS_DM_00164] [SWS_DM_00217] [SWS_DM_00218] [SWS_DM_00229] [SWS_DM_00244] [SWS_DM_00245] [SWS_DM_00246] [SWS_DM_00247] [SWS_DM_00370] [SWS_DM_00371] [SWS_DM_00372] [SWS_DM_00373] [SWS_DM_00374] [SWS_DM_00909] [SWS_DM_00910]
[RS_Diag_04182]	Provide an application interface to change operation cycles states	[SWS_DM_00756] [SWS_DM_00885]
[RS_Diag_04183]	Notify interested parties about event status changes	[SWS_DM_00650] [SWS_DM_00886] [SWS_DM_00887]
[RS_Diag_04185]	Notify applications about the clearing of an event	[SWS_DM_00562]
[RS_Diag_04186]	Notify applications about the start or restart of an operation cycle	[SWS_DM_00563] [SWS_DM_00755]
[RS_Diag_04189]	Support a fine grained configuration for Snapshot Records and ExtendedData Records	[SWS_DM_00151] [SWS_DM_00155]
[RS_Diag_04190]	Usage of internal data elements in SnapshotRecords and ExtendedDataRecords	[SWS_DM_00017] [SWS_DM_00030] [SWS_DM_00152] [SWS_DM_00154]
[RS_Diag_04192]	Provide the ability to handle event specific enable conditions	[SWS_DM_00564] [SWS_DM_00568] [SWS_DM_00881] [SWS_DM_00882]
[RS_Diag_04194]	ClearDTC shall be accessible for applications	[SWS_DM_00262] [SWS_DM_00671] [SWS_DM_00897] [SWS_DM_00898] [SWS_DM_00899] [SWS_DM_00900] [SWS_DM_00901]
[RS_Diag_04195]	Chronological reporting order of the DTCs located in the configured event memory	[SWS_DM_NA]

Requirement	Description	Satisfied by
[RS_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_00113] [SWS_DM_00114] [SWS_DM_00126] [SWS_DM_00127] [SWS_DM_00128] [SWS_DM_00134] [SWS_DM_00137] [SWS_DM_00140] [SWS_DM_00141] [SWS_DM_00162] [SWS_DM_00170] [SWS_DM_00186] [SWS_DM_00199] [SWS_DM_00201] [SWS_DM_00227] [SWS_DM_00234] [SWS_DM_00235] [SWS_DM_00236] [SWS_DM_00269] [SWS_DM_00360] [SWS_DM_00361] [SWS_DM_00363] [SWS_DM_00376] [SWS_DM_00571] [SWS_DM_00573] [SWS_DM_00574] [SWS_DM_00575] [SWS_DM_00576] [SWS_DM_00860] [SWS_DM_00861] [SWS_DM_00866] [SWS_DM_00867]
[RS_Diag_04197]	Clearing the user defined fault memory	[SWS_DM_00193] [SWS_DM_00194] [SWS_DM_00195] [SWS_DM_00208]
[RS_Diag_04198]	Process all UDS Services related to session and security management of ISO 14229 internally	[SWS_DM_00226] [SWS_DM_00228]
[RS_Diag_04199]	Provide a configurable UDS service execution mechanism at runtime to decide if a UDS request shall be processed or not	[SWS_DM_00111] [SWS_DM_00112] [SWS_DM_00286] [SWS_DM_00287] [SWS_DM_00288] [SWS_DM_00289] [SWS_DM_00770] [SWS_DM_00771] [SWS_DM_00772] [SWS_DM_00773] [SWS_DM_00774] [SWS_DM_00775] [SWS_DM_00776] [SWS_DM_00777] [SWS_DM_00857] [SWS_DM_00858]
[RS_Diag_04200]	Support event combination	[SWS_DM_NA]
[RS_Diag_04201]	Support a configuration to assign specific events to a customer specific DTC	[SWS_DM_00060] [SWS_DM_00642] [SWS_DM_00653] [SWS_DM_CONSTR_00059]
[RS_Diag_04202]	Report DTCs getting active to the error logging module/system	[SWS_DM_NA]
[RS_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_00252] [SWS_DM_00409] [SWS_DM_00412] [SWS_DM_00413] [SWS_DM_00414] [SWS_DM_00415] [SWS_DM_00416] [SWS_DM_00417] [SWS_DM_00437] [SWS_DM_00438] [SWS_DM_00439] [SWS_DM_00440] [SWS_DM_00441]

Requirement	Description	Satisfied by
		[SWS_DM_00442] [SWS_DM_00443] [SWS_DM_00444] [SWS_DM_00445] [SWS_DM_00446] [SWS_DM_00447] [SWS_DM_00448] [SWS_DM_00450] [SWS_DM_00507] [SWS_DM_00863] [SWS_DM_00864]
[RS_Diag_04204]	Provide the current status of each warning indicator.	[SWS_DM_00221] [SWS_DM_00223] [SWS_DM_00224] [SWS_DM_00651] [SWS_DM_00740] [SWS_DM_00741] [SWS_DM_00742] [SWS_DM_00743] [SWS_DM_00744] [SWS_DM_00745] [SWS_DM_00888]
[RS_Diag_04205]	Support of SnapshotRecords	[SWS_DM_00151] [SWS_DM_00152] [SWS_DM_00660] [SWS_DM_00661] [SWS_DM_00662] [SWS_DM_00668] [SWS_DM_00893]
[RS_Diag_04206]	Support of ExtendedData Records	[SWS_DM_00154] [SWS_DM_00155] [SWS_DM_00895]
[RS_Diag_04208]	Inform the application about diagnostic session and diagnostic security level changes on each tester connection.	[SWS_DM_00270] [SWS_DM_00271] [SWS_DM_00272] [SWS_DM_00478] [SWS_DM_00479] [SWS_DM_00480] [SWS_DM_00845] [SWS_DM_00846] [SWS_DM_CONSTR_00208]
[RS_Diag_04211]	Persistent storage of DTC status and environmental data	[SWS_DM_00148] [SWS_DM_00150]
[RS_Diag_04214]	Support the user defined fault memories defined by ISO 14229-1	[SWS_DM_00055] [SWS_DM_00057] [SWS_DM_00083] [SWS_DM_00911] [SWS_DM_CONSTR_00084]
[RS_Diag_04216]	Support for multiple Diagnostic Server Instances	[SWS_DM_00390] [SWS_DM_00391] [SWS_DM_00420]
[RS_Diag_04218]	Support of UDS service 0x2F InputOutputControlByIDentifier	[SWS_DM_NA]
[RS_Diag_04224]	Support the UDS service 0x31 (RoutineControl) according to ISO 14229-1	[SWS_DM_00201] [SWS_DM_00202] [SWS_DM_00203] [SWS_DM_00437] [SWS_DM_00448] [SWS_DM_00551] [SWS_DM_00552] [SWS_DM_00553] [SWS_DM_00554] [SWS_DM_00555] [SWS_DM_00556] [SWS_DM_00557] [SWS_DM_00574] [SWS_DM_00575] [SWS_DM_00576] [SWS_DM_00591] [SWS_DM_00592] [SWS_DM_00593] [SWS_DM_00594] [SWS_DM_00605]
[RS_Diag_04225]	The diagnostic in AUTOSAR shall support event specific time base debounce counters	[SWS_DM_00015] [SWS_DM_00030] [SWS_DM_00032] [SWS_DM_00033] [SWS_DM_00035] [SWS_DM_00036] [SWS_DM_00038] [SWS_DM_00039] [SWS_DM_00040] [SWS_DM_00085] [SWS_DM_00086] [SWS_DM_00539] [SWS_DM_00550] [SWS_DM_00645] [SWS_DM_00654] [SWS_DM_00877] [SWS_DM_00878] [SWS_DM_00879] [SWS_DM_00880]

Requirement	Description	Satisfied by
[RS_Diag_04242]	The DoIP module shall support Vehicle Internal Testers.	[SWS_DM_00815] [SWS_DM_00816] [SWS_DM_00820] [SWS_DM_00821] [SWS_DM_00822] [SWS_DM_00830] [SWS_DM_00831] [SWS_DM_00832] [SWS_DM_00833] [SWS_DM_00834] [SWS_DM_00835] [SWS_DM_00836] [SWS_DM_00837]
[SRS_Diag_04180]	No description	[SWS_DM_00165]
[SRS_Eth_00026]	No description	[SWS_DM_00449] [SWS_DM_00720] [SWS_DM_00721] [SWS_DM_00722] [SWS_DM_00723] [SWS_DM_00724] [SWS_DM_00725] [SWS_DM_00726] [SWS_DM_00813] [SWS_DM_00814] [SWS_DM_00855] [SWS_DM_CONSTR_00206]
[SRS_Eth_00027]	No description	[SWS_DM_00449]
[SRS_Eth_00080]	No description	[SWS_DM_00449] [SWS_DM_00730] [SWS_DM_00731] [SWS_DM_00732] [SWS_DM_00733] [SWS_DM_00734] [SWS_DM_00735] [SWS_DM_00736] [SWS_DM_00814]
[SRS_Eth_00082]	No description	[SWS_DM_00449]
[SRS_Eth_00083]	No description	[SWS_DM_00005] [SWS_DM_00449]
[SRS_Eth_00084]	No description	[SWS_DM_00449]

6.1 Not applicable requirements

[SWS_DM_NA]{DRAFT} ⌈These requirements are not applicable as they are not within the scope of this release.⌋ *(RS_Diag_04059, RS_Diag_04064, RS_Diag_04171, RS_Diag_04195, RS_Diag_04200, RS_Diag_04202, RS_Diag_04218)*

7 Functional specification

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

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

The link between the UDS Transport Layer and the Application Layer is implemented by the Transport Protocol Manager (see subsection 8.1.4 “UdsTransportProtocolMgr Class”.), which dispatches UDS messages in both directions: UDS requests from Diagnostic Clients are forwarded to the respective responsible Diagnostic Server Instance, and UDS responses created by Diagnostic Server Instance are dispatched towards the respective Transport Protocol Handler (see subsection 8.1.3 “UdsTransportProtocolHandler Class”.) that handles the connection to the Diagnostic Client.

A broad subcomponent view on DM is given as follows:

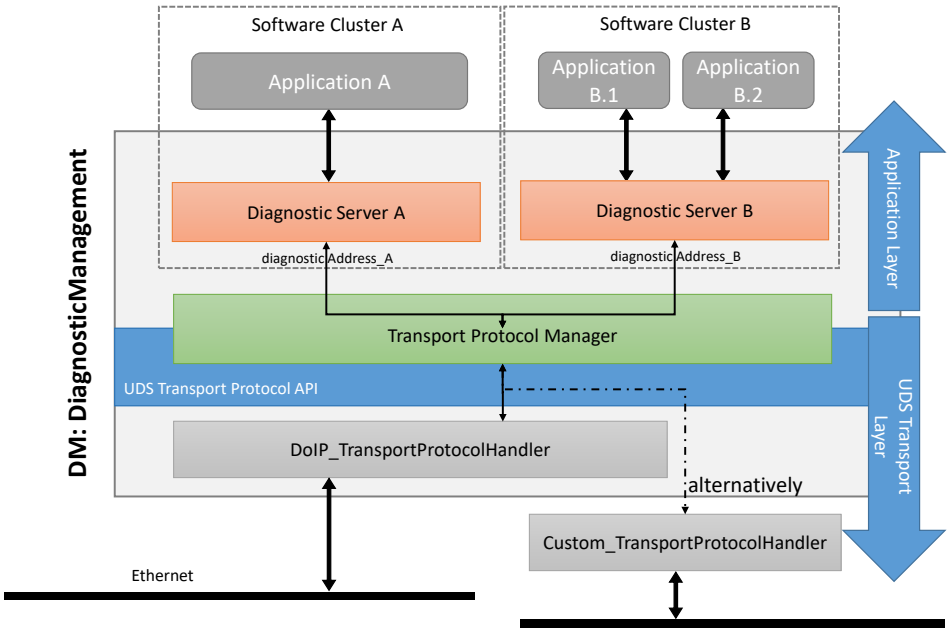

Figure 7.1: Component view on Diagnostic Management

7.1 UDS Transport Layer

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

7.1.1 Support of proprietary UDS Transport Layer

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

7.1.1.1 Initialization, Starting and Stopping of a proprietary UDS TransportLayer

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

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

⌋*(RS_Diag_04168)*

[SWS_DM_00330]{DRAFT} **Construction of an `Uds Transport Protocol` implementation** ⌈The `DM` shall call the specific constructor of the `Uds Transport Protocol` implementation, where the argument `handler_id` is unique among all by `DM` instantiated `Uds Transport Protocol` implementations and the `transport_protocol_mgr` is set to the reference of the instance of `UdsTransportProtocolMgr` (see [SWS_DM_00306]) provided by `DM`.⌋*(RS_Diag_04168)*

[SWS_DM_00331]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00332]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00333]{DRAFT} **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.⌋(RS_Diag_04168)

[SWS_DM_00340]{DRAFT} **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.⌋(RS_Diag_04168)

7.1.1.2 UDS message reception on a proprietary UDS TransportLayer

[SWS_DM_00342]{DRAFT} **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

⌋(RS_Diag_04168)

[SWS_DM_00347]{DRAFT} **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.⌋(RS_Diag_04168)

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

- The UDS source address (SA) in the clients/testers request which represent a technology/transport layer independent part.
- The technology/transport layer specific/dependent network endpoint source address, from which the request from the client originates. In Ethernet-based networks this typically is an IP-address/port number pair, while in CAN networks it is the CAN identifier of the CAN-TP message used by the client. In UDS on CAN (ISO ISO-15765-2[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 Diagnostic Server, which supports parallel Diagnostic Clients, it is a perfectly valid scenario that two Diagnostic Clients with the same UDS SA can be active in parallel if they originate from different/distinguishable network endpoints.

[SWS_DM_00385]{DRAFT} **Acceptance of UDS message reception** ⌈If the DM is able to process the indicated request, it shall return a std::pair with IndicationResult set to kIndicationOk and a UdsMessagePtr, which owns a valid UdsMessage object, with a capacity of so many bytes, the DM wants to process of the indicated request. The minimum size of the UdsMessage object shall be one byte.⌋ *(RS_Diag_04168)*

Note: For details about std::pair see [SWS_DM_00309].

[SWS_DM_00392]{DRAFT} **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).⌋*(RS_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]{DRAFT} **Ignoring UDS message reception because DM is busy** ⌈If the DM is busy and not able to process the indicated UDS request, it shall return a std::pair with IndicationResult set to kIndicationOccupied and a UdsMessagePtr equal to UdsMessagePtr(nullptr).⌋*(RS_Diag_04168)*

Note: For details about std::pair see [SWS_DM_00309].

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]{DRAFT} **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).⌋*(RS_Diag_04168)*

Note: For details about std::pair see [SWS_DM_00309].

Note: There might exist Uds Transport Protocol implementations, which make NO distinction between [SWS_DM_00386] and [SWS_DM_00387]. I.e. regardless, whether the DM returns a kIndicationOverflow or kIndicationOccupied, the

behavior on transport layer level is the same. But, for instance, a CanTP Uds Transport Protocol implementation, would explicitly react on a kIndicationOverflow with sending a FC.OFLW on CanTP level to the UDS request sender.

[SWS_DM_00487]{DRAFT} **Ignoring UDS message reception because of unknown target address** ⌈If the DM is not able to process the indicated UDS request, because the indicated target address is unknown to DM, it shall return a std::pair with IndicationResult set to kIndicationUnknownTargetAddress and a UdsMessagePtr equal to UdsMessagePtr(nullptr).⌋*(RS_Diag_04168)*

Note: For details about std::pair see [SWS_DM_00309].

[SWS_DM_00388]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00345]{DRAFT} **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.⌋ *(RS_Diag_04168)*

[SWS_DM_00389]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00346]{DRAFT} **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.⌋ *(RS_Diag_04168)*

7.1.1.3 UDS message transmission on a proprietary UDS TransportLayer

[SWS_DM_00348]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00349]{DRAFT} **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]).⌋*(RS_Diag_04168)*

[SWS_DM_00350]{DRAFT} **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]).⌋*(RS_Diag_04168)*

[SWS_DM_00351]{DRAFT} **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.⌋*(RS_Diag_04168)*

7.1.1.4 Channel Notifications

Each incoming UDS request message is assigned an exact Uds Transport Protocol implementation specific Channel. With the normal request/reply paradigm in diagnostics, the UDS response message is sent out at the same Channel, from which the UDS request has been received. Therefore the Channel identifier is given to the DM in IndicateMessage (see [SWS_DM_00309]) in the form of parameter global_channel_id. The Channel part from this parameter is then used in the corresponding response in Transmit (see [SWS_DM_00327]).

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

[SWS_DM_00356]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00357]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00358]{DRAFT} **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.⌋*(RS_Diag_04168)*

[SWS_DM_00359]{DRAFT} **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.⌋*(RS_Diag_04168)*

7.1.2 DoIP

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

[SWS_DM_00475]{DRAFT} **DoIP Version** ⌈DM shall support following version of the DoIP ISO 13400-2 specification: 2020.⌋*()*

Note: According to the ISO 13400-2[11] specification, the DoIP entity shall support protocol version = 0xFF in the vehicle identification request message.

[SWS_DM_00449]{DRAFT} **Supported DoIP message types** ⌈DM shall support the DoIP message types listed in Table 7.1.⌋*(SRS_Eth_00026, SRS_Eth_00080, SRS_Eth_00082, SRS_Eth_00083, SRS_Eth_00084, SRS_Eth_00027)*

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

Table 7.1: Supported DoIp message types

[SWS_DM_00855]{DRAFT} **Providing the VIN in DoIP protocol messages** ⌈If the DM needs to know VIN to be able to react or answer on some DoIP messages, it shall obtain it according to [SWS_DM_00903].⌋*(SRS_Eth_00026)*

[SWS_DM_00814]{DRAFT} **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 calling the method

ara::diag::DoIPPowerMode::GetDoIPPowerMode() ([SWS_DM_00734]).⌋*(SRS_Eth_00026, SRS_Eth_00080)*

[SWS_DM_00813]{DRAFT} **Providing the GID in DoIP protocol messages** ⌈If the DM needs to know the GID and the status of the GID to be able to react or answer on any DoIP message, it shall obtain it by calling the method *ara::diag::DoIPGroupIdentification::GetGidStatus()* ([SWS_DM_00724]).⌋*(SRS_Eth_00026)*

[SWS_DM_00815]{DRAFT} **When to send Vehicle announcement messages on interfaces without activation line control** ⌈The DM gets notified, when to send out vehicle announcement messages on a network interface without activation line control (isActivationLineDependent == FALSE) by a call to method *diag::DoIPTriggerVehicleAnnouncement::TriggerVehicleAnnouncement()* ([SWS_DM_00822]), which DM has to provide. The method call contains the network interface identified via networkInterfaceId on which the announcement shall be sent.⌋*(RS_Diag_04242)*

[SWS_DM_00816]{DRAFT} **Notification of activation line status change on activation line controlled network interfaces** ⌈The DM gets notified, when the activation line status changes for activation line controlled network interfaces (isActivationLineDependent == TRUE) via software components providing an instance of DiagnosticDoIPActivationLineInterface. The DM shall identify for which network interface an instance of DiagnosticDoIPActivationLineInterface is providing the activation line status via call to method *diag::DoIPTriggerVehicleAnnouncement::GetNetworkInterfaceId()* ([SWS_DM_00833]). Whenever the status of the activation line of the related network interface changes, the application calls *diag::DoIPTriggerVehicleAnnouncement::UpdateActivationLineState()* ([SWS_DM_00834]).⌋*(RS_Diag_04242)*

7.1.3 Dispatching of UDS Requests

The Transport Protocol Manager has to dispatch the UDS-messages between the Transport Protocol Handler and the Diagnostic Server instances. To do this the Transport Protocol Manager uses the following information as provided by the Transport Protocol Handler indication function on received UDS requests:

- Target Address
- Target Address Type (phys / func)

In transmit direction the Transport Protocol Manager provides the UDS message from the Diagnostic Server and calls the Transmit method from the Transport Protocol Handler.

[SWS_DM_00390]{DRAFT} **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.⌋*(RS_Diag_04216)*

[SWS_DM_00391]{DRAFT} **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.⌋*(RS_Diag_04216)*

7.2 Diagnostic Server

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

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

[SWS_DM_00420]{DRAFT} **Instantiation of Diagnostic Server** ⌈DM shall instantiate an independent Diagnostic Server per configured SoftwareCluster which references a DiagnosticContributionSet in the role of diagnosticExtract with dedicated resources and functionality configured by this DiagnosticContributionSet.⌋*(RS_Diag_04216)*

Details on required configuration items are described in section 7.2.3.

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

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

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

7.2.1 Diagnostic Communication Management

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

7.2.1.1 Diagnostic Conversations

A Diagnostic Conversation depicts a conversation between a distinct Diagnostic Client and a Diagnostic Server instance. In contrast to CP, on AP the details of connections between Diagnostic Clients and Diagnostic Server instances are not statically configured, but a Diagnostic Conversation is dynamically allocated during run-time of the Diagnostic Server instance.

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

[SWS_DM_00421]{DRAFT} **Identification of a Diagnostic Client** ⌈The Diagnostic Server instance shall identify a Diagnostic Client by means of the tuple of source_addr and gobal_channel_id provided by the TP Layer on call of IndicateMessage, see [SWS_DM_00347].⌋*(RS_Diag_04005)*

[SWS_DM_00046]{DRAFT} **Each Diagnostic Conversation has its own session resources** ⌈The Diagnostic Server instance shall provide each Diagnostic Conversation with its own and independently managed diagnostic session, which can be any valid UDS session type.⌋*(RS_Diag_04119, RS_Diag_04006)*

[SWS_DM_00047]{DRAFT} **Each Diagnostic Conversation has its own security-level resources** ⌈The Diagnostic Server instance shall provide each Diagnostic Conversation with its own and independently managed security-level.⌋*(RS_Diag_04005)*

7.2.1.1.1 Parallel Client Handling Variants

There are generally various approaches for a server (which the Diagnostic Server instance 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.

Pseudo Parallel Mode The characteristic of this parallelism mode is, that there is only a real parallelism as long as no Diagnostic Client switches to a non-default session. At the point in time one Diagnostic Client has switched to a non-default session, requests of other diagnostic clients (other Diagnostic Conversations) get rejected with the exception if the newly requested Diagnostic Conversation has a higher priority than the current Diagnostic Conversation in non-default session. This characteristic of the 'pseudo parallel mode' means, that the diagnostic session state is not an individual state per Diagnostic Client, but it becomes a **global state for the entire Diagnostic Server instance**.

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

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

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

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

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

7.2.1.1.2 Life-cycle of a Diagnostic Conversation

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

- UDS request processing is finished by either
  - sending positive or final negative response and processing TransmitConfirmation ([SWS_DM_00312]) call from TP-layer according to [SWS_DM_00350],
  - suppressing positive response according to [SWS_DM_00365],
  - suppressing negative response according to [SWS_DM_00862].
  - suppressing any response according to [SWS_DM_00860].
- associated Session is the Default Session.

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

7.2.1.1.3 Diagnostic Conversation Service Interface

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

[SWS_DM_00840]{DRAFT} **Instantiation of Diagnostic Conversation Interface** ⌈The `Diagnostic Server instance` shall provide as many instances of `diag::Conversation class` ([SWS_DM_00693])as the number of potential parallel `Diagnostic Clients` is configured by `maxTesterConnections`.⌋*(RS_Diag_04166)*

[SWS_DM_00841]{DRAFT} **Assignment of Diagnostic Conversation to Service Instances** ⌈On establishment of a new `Diagnostic Conversation`, the `Diagnostic Server instance` shall assign this `Diagnostic Conversation` to an inactive `diag::Conversation class` Instance, i.e. the field value of `diag::Conversation::ActivityStatusType` is set to `kInactive`. After assignment, the fields of the `diag::Conversation class` Instance shall be updated according to the state of the given `Diagnostic Conversation`, i.e.,

- `diag::Conversation::ActivityStatusType` set to `kActive`,
- `diag::Conversation::ConversationIdentifierType` matching the values of `udstransport::UdsTransportProtocolMgr::IndicateMessage` ([SWS_DM_00309]) call, that initiated the creation of this `Diagnostic Conversation` (see [SWS_DM_00347]),
- a call to `diag::Conversation::GetDiagnosticSession` ([SWS_DM_00696]) will return the Diagnostic Session of this `Diagnostic Conversation`,

- a call to diag::Conversation::GetDiagnosticSecurityLevel ([SWS_DM_00698]) will return the Diagnostic Security Level of this Diagnostic Conversation.

⌋(*RS_Diag_04166*)

[SWS_DM_00844]{DRAFT} **Updating DiagnosticConversation Service Instance fields** ⌈During the life-cycle of a Diagnostic Conversation, the Diagnostic Server instance shall update the fields of the assigned ara::diag::Conversation class instance according to any change of the state of the Diagnostic Conversation.⌋(*RS_Diag_04166*)

[SWS_DM_00843]{DRAFT} **Reset Service Instance fields on end of Diagnostic Conversation** ⌈If the life-cycle of a Diagnostic Conversation ends, the Diagnostic Server instance shall reset the field values of the assigned diag::Conversation class Instance to its predefined initial values.⌋(*RS_Diag_04166*)

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

[SWS_DM_00842]{DRAFT} **Default session change trigger from AAs** ⌈If diag::Conversation::ResetToDefaultSession method is called, the Diagnostic Server instance shall complete the latest ongoing request and then switch the Diagnostic Session of this Diagnostic Conversation to Default Session.⌋(*RS_Diag_04006*)

7.2.1.2 Assignment of UDS requests to Diagnostic Conversations

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

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

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

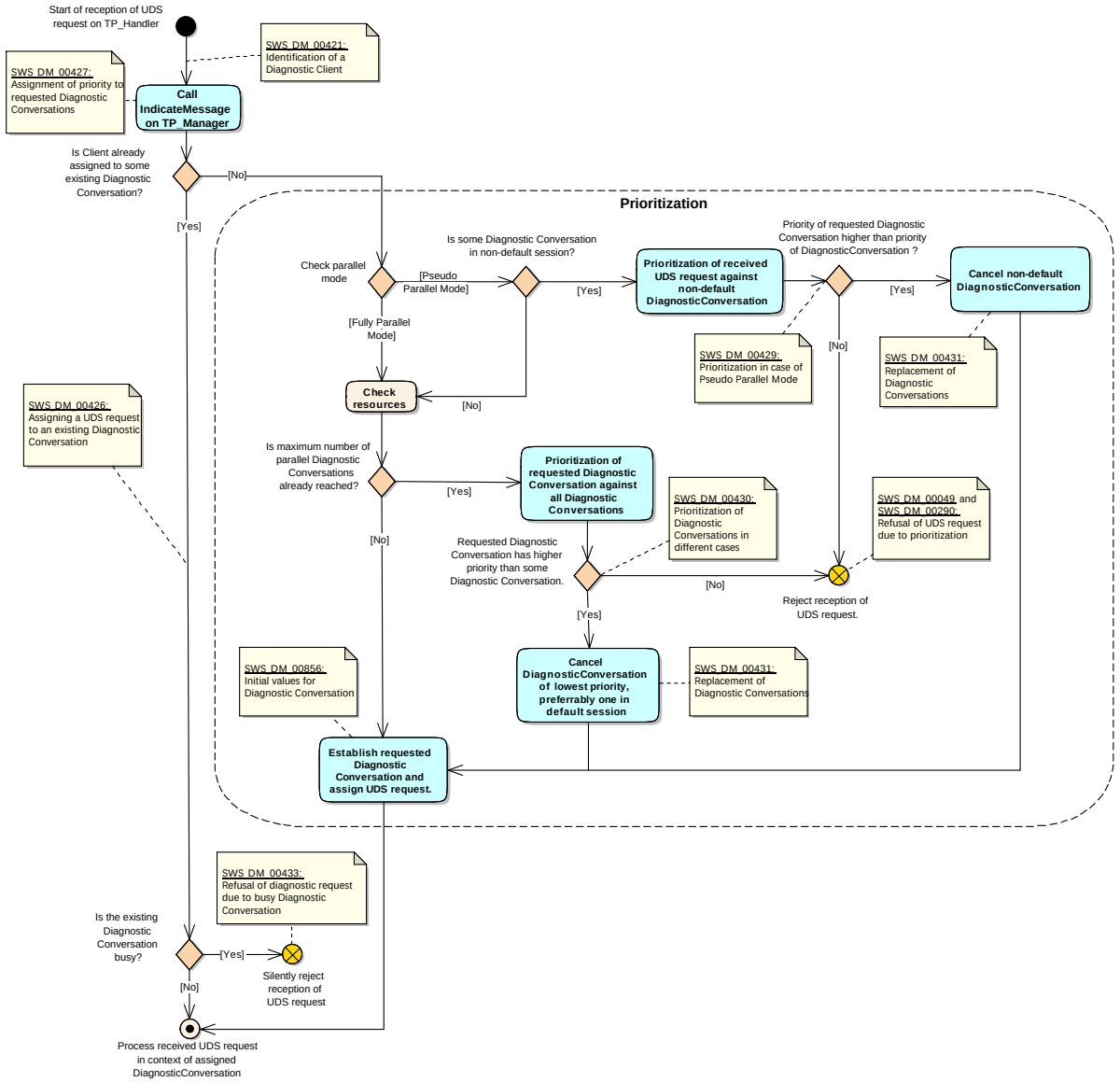

Figure 7.2: UDS request assignment to a Diagnostic Conversation and Prioritization

[SWS_DM_00425]{DRAFT} **Procedure to assign UDS requests to Diagnostic Conversations** ⌈The `Diagnostic Server instance` shall handle a newly received UDS request as specified in Figure 7.2.⌋*(RS_Diag_04166)*

[SWS_DM_00426]{DRAFT} **Assigning a UDS request to an existing Diagnostic Conversation** ⌈If a UDS request is received and there already exists a Diagnostic Conversation associated to the transmitting Diagnostic Client, then the `Diagnostic Server instance` shall assign this UDS request to the same `Diagnostic Conversation`.⌋*(RS_Diag_04166)*

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

7.2.1.2.1 Prioritization

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

[SWS_DM_00427]{DRAFT} **Priority of a Diagnostic Conversation** ⌈The Diagnostic Server instance shall take as the priority of a Diagnostic Conversation the respective value provided by IndicateMessage call according to [SWS_DM_00309] that established the Diagnostic Conversation.⌋*(RS_Diag_04166)*

[SWS_DM_00428]{DRAFT} **Treatment of priority values** ⌈The Diagnostic Server instance shall consider a lower value as higher priority and vice versa. In particular, priority value 0 represents highest priority.⌋*(RS_Diag_04166)*

[SWS_DM_00429]{DRAFT} **Prioritization in active non-default session** ⌈If a Diagnostic Conversation is in non-default session, the Diagnostic Server shall compare the priority of the requested Diagnostic Conversation against the priority of the given Diagnostic Conversation in non-default session. If the priority of the requested Diagnostic Conversation is higher than the priority of the Diagnostic Conversation in non-default Session, the Diagnostic Server instance shall replace the Diagnostic Conversation in non- default session by the requested Diagnostic Conversation according to [SWS_DM_00431] and assign the UDS request to the newly established Diagnostic Conversation.⌋*(RS_Diag_04166)*

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

- If all priorities of the existing Diagnostic Conversations are higher or equal to the priority of the requested Diagnostic Conversation, the Diagnostic Server instance shall refuse the UDS request according to [SWS_DM_00049] and [SWS_DM_00290].
- If some priority of the existing Diagnostic Conversations is lower than the priority of the requested Diagnostic Conversation, the Diagnostic Server instance shall replace the Diagnostic Conversation of lowest priority by the requested Diagnostic Conversation according to [SWS_DM_00431] and assign the UDS request to the newly established Diagnostic Conversation. If multiple Diagnostic Conversations exist with the same lowest priority, the Diagnostic Server instance shall prefer replacement of a Diagnostic Conversation within default Session before replacement of a Diagnostic Conversation in non-default Session.

⌋(*RS_Diag_04166*)

7.2.1.2.2 Replacement of Diagnostic Conversations and initial values

[SWS_DM_00431]{DRAFT} **Replacement of Diagnostic Conversations** ⌈On replacement of a given `Diagnostic Conversation` by a requested `Diagnostic Conversation`, the `Diagnostic Server instance` shall cancel the given `Diagnostic Conversation` according to [SWS_DM_00482] and establish a new `Diagnostic Conversation` as requested.⌋(*RS_Diag_04167*)

[SWS_DM_00856]{DRAFT} **Initial values for Diagnostic Conversation** ⌈For a newly established Diagnostic Conversation, the `Diagnostic Server instance` shall use the following initial values:

- Session set to Default Session, which is synonymous with returning an according ara::core::StringView when `diag::Conversation::GetDiagnosticSession()` ([SWS_DM_00696]) is called and
- Security Level set to status Locked, which is synonymous with returning an according ara::core::StringView when `diag::Conversation::GetDiagnosticSecurityLevel()` ([SWS_DM_00698]) is called .

⌋(*RS_Diag_04166*)

7.2.1.2.3 Refusal of incoming diagnostic request

[SWS_DM_00433]{DRAFT} **Refusal of diagnostic request due to busy Diagnostic Conversation** ⌈If a UDS request is assigned to a `Diagnostic Conversation` that has not finished processing of a formerly assigned UDS request, then the `Diagnostic Server instance` shall ignore the new UDS request according to [SWS_DM_00386].⌋(*RS_Diag_04020*)

[SWS_DM_00049]{DRAFT} **Refusal of diagnostic request due to prioritization with BusyRepeatRequest** ⌈If prioritization demands refusal of an incoming UDS request and the configuration parameter `DiagnosticCommonProps.responseOnSecondDeclinedRequest` is TRUE, the `Diagnostic Server instance` shall accept this request according to [SWS_DM_00385] without further processing and a negative response with NRC 0x21 (BusyRepeatRequest) shall be issued for this request.⌋(*RS_Diag_04167*)

[SWS_DM_00290]{DRAFT} **Refusal of diagnostic request due to prioritization without response** ⌈If prioritization demands refusal of an incoming UDS request and the configuration parameter `DiagnosticCommonProps.responseOnSecondDeclinedRequest` is FALSE, the `Diagnostic Server instance` shall ignore

this request according to [SWS_DM_00386] without further processing and no response shall be issued.⌋(*RS_Diag_04167*)

7.2.1.3 UDS request Validation/Verification

[SWS_DM_00096]{DRAFT} **Validation Steps and Order** ⌈The `Diagnostic Server instance` shall execute the request validation, negative response code determination and processing according to ISO 14229-1[1].⌋(*RS_Diag_04196*, *RS_Diag_04203*)

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

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

[SWS_DM_00097]{DRAFT} **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.⌋(*RS_Diag_04196*)

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

7.2.1.3.1 UDS request format checks

[SWS_DM_00098]{DRAFT} **UDS message checks** ⌈The `Diagnostic Server instance` 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)⌋(*RS_Diag_04203*)

7.2.1.3.2 Supported service checks

[SWS_DM_00099]{DRAFT} **Supported Service SID level checks** ⌈The Diagnostic Server instance 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)⌋*(RS_Diag_04203)*

[SWS_DM_00100]{DRAFT} **Supported Service subfunction level checks** ⌈The Diagnostic Server instance 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)⌋*(RS_Diag_04203)*

7.2.1.3.3 Session and Security Checks

[SWS_DM_00101]{DRAFT} **Session Access SID level Permission** ⌈The Diagnostic Server instance 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)⌋*(RS_Diag_04203, RS_Diag_04006)*

[SWS_DM_00102]{DRAFT} **Session Access subfunction level Permission** ⌈The Diagnostic Server instance 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)⌋*(RS_Diag_04203, RS_Diag_04006)*

[SWS_DM_00103]{DRAFT} **Security Access level Permission** ⌈The `Diagnostic Server instance` 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).⌋*(RS_Diag_04203, RS_Diag_04005)*

[SWS_DM_00450]{DRAFT} **Security Access subfunction level Permission** ⌈The `Diagnostic Server instance` 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 and:

- the `SID` of the service has subfunctions
- and at least one other sub-functions has access permissions in the current Security Level,

the Verification shall be considered as failed and the negative response code shall be 0x33 (securityAccessDenied)⌋*(RS_Diag_04203)*

7.2.1.3.4 Manufacturer and Supplier Permission Checks and Confirmation

[SWS_DM_00857]{DRAFT} **Signature of Manufacturer Permission Check Method** ⌈The `Diagnostic Server instance` shall call `diag::ServiceValidation::Validate()` ([SWS_DM_00774]) on each received request message. In case a call returned an error, the Verification shall be considered as failed and the negative response code shall be equal to the value of the error code according to [SWS_DM_00547].⌋*(RS_Diag_04199)*

[SWS_DM_00858]{DRAFT} **Signature of Supplier Permission Check Method** ⌈The `Diagnostic Server instance` shall call `diag::ServiceValidation::Validate()` ([SWS_DM_00774]) on each received request message. In case a call returned an error, the Verification shall be considered as failed and the negative response code shall be equal to the value of the error code according to [SWS_DM_00547].⌋*(RS_Diag_04199)*

[SWS_DM_00859]{DRAFT} **Confirmation of service processing** ⌈The `Diagnostic Server instance` shall call the method `diag::ServiceValidation::Confirmation()` ([SWS_DM_00775]) on every service instances for which `diag::ServiceValidation::Validate()` ([SWS_DM_00774]) was called. If message handling results in sending a positive or negative response, the `diag::ServiceValidation::Confirmation()` ([SWS_DM_00775]) call shall be deferred after reception of TransmitConfirmation

([SWS_DM_00312]). In any other case, it shall be the last step of request processing.⌋ *(RS_Diag_04019, RS_Diag_04172)*

[SWS_DM_00860]{DRAFT} **No service processing** ⌈If Manufacturer- or Supplier Permission Check (according to [SWS_DM_00857] or [SWS_DM_00858]) returns the error code `kNoProcessingNoResponse`, the `Diagnostic Server instance` shall call without any service processing the `diag::ServiceValidation::Confirmation()` ([SWS_DM_00775]) with `diag::ServiceValidation::ConfirmationStatusType status` parameter set to `kNoProcessingNoResponse` and do no response message.⌋ *(RS_Diag_04196)*

7.2.1.3.5 Condition checks

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

[SWS_DM_00111]{DRAFT} **Configurable environment condition checks** ⌈The `Diagnostic Server instance` shall perform a condition check when the ISO 14229-1[1] mentions a service specific “Condition check” in the defined NRC handling for a given diagnostic service. The `Diagnostic Server instance` shall send the configured NRC value (see [SWS_DM_00289]) if the condition is not fulfilled.⌋ *(RS_Diag_04199)*

[SWS_DM_00112]{DRAFT} **Condition check definition** ⌈The `Diagnostic Server instance` 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`.⌋ *(RS_Diag_04199)*

[SWS_DM_00286]{DRAFT} **Configurable environmental condition check execution** ⌈The `Diagnostic Server instance` shall execute an environmental condition check before executing the requested service if defined. (see `DiagnosticEnvironmentalCondition` element from DEXT [2]).⌋ *(RS_Diag_04199)*

[SWS_DM_00287]{DRAFT} **Configurable environmental condition check criteria** ⌈The environmental condition check shall be done by evaluation of the configured `DiagnosticEnvConditionFormula`.⌋ *(RS_Diag_04199)*

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

[SWS_DM_00288]{DRAFT} **Configurable environmental condition check evaluates to `TRUE`** ⌈If the computation of the `DiagnosticEnvConditionFormula` evaluated to `TRUE`, the `Diagnostic Server instance` shall execute the requested service.⌋ *(RS_Diag_04199)*

[SWS_DM_00289]{DRAFT} **Configurable environmental condition check evaluates to `FALSE`** ⌈The `Diagnostic Server instance` shall send the NRC defined in `nrcValue`, if the computation of the `DiagnosticEnvConditionFormula` evaluated to `FALSE`. If `nrcValue` does not define a NRC, the `Diagnostic Server instance` shall send NRC 0x22 (ConditionsNotCorrect).⌋*(RS_Diag_04199)*

7.2.1.4 UDS response handling

7.2.1.4.1 Positive and negative responses

[SWS_DM_00376]{DRAFT} **Positive response processing** ⌈If an external service processor did not raise an `ApApplicationError`, the `Diagnostic Server instance` shall return a positive response.⌋*(RS_Diag_04196)*

[SWS_DM_00861]{DRAFT} **Negative response processing** ⌈If the external processor raised an error according to [SWS_DM_00547], the `Diagnostic Server instance` shall return a negative response with the value of the error code. For details see ISO 14229-1[1]; chapter 10.2.⌋*(RS_Diag_04196)*

7.2.1.4.2 Suppression of responses

[SWS_DM_00365]{DRAFT} **Suppression of positive response in accordance to ISO 14229-1[1]** ⌈In the case that the "suppressPosRspMsgIndicationBit" is set in the request, the `Diagnostic Server instance` shall suppress the positive response.⌋*(RS_Diag_04020)*

[SWS_DM_00862]{DRAFT} **Suppression of negative response for functional requests in accordance to ISO 14229-1[1]** ⌈If the external processor raised an error according to [SWS_DM_00547], the `Diagnostic Server instance` shall suppress a negative response for the following error codes:

- `kServiceNotSupported` ([SWS_DM_00526]),
- `kSubfunctionNotSupported` ([SWS_DM_00526]),
- `kRequestOutOfRange` ([SWS_DM_00526]),
- `kServiceNotSupportedInActiveSession` ([SWS_DM_00526]) or
- `kSubFunctionNotSupportedInActiveSession` ([SWS_DM_00526])

and the request is functional addressed.⌋*(RS_Diag_04020)*

7.2.1.4.3 Sending busy Responses

[SWS_DM_00368]{DRAFT} **Sending busy responses** ⌈If the `Diagnostic Server instance` is able to perform a diagnostic service, but needs additional time to finish the task and prepare the response, then the `Diagnostic Server instance` shall send a negative response with NRC 0x78 (Response pending) when reaching the response time (`p2ServerMax`/`p2StarServerMax`).⌋*(RS_Diag_04016)*

[SWS_DM_00369]{DRAFT} **Maximum number of busy responses** ⌈If the number of negative responses for a requested diagnostic request reaches the value defined in the configuration parameter `maxNumberOfRequestCorrectlyReceivedResponsePending`, the `Diagnostic Server instance` module shall cancel the processing the active diagnostic request (according to [SWS_DM_00482]) and send a negative response with NRC 0x10 (General reject).⌋*(RS_Diag_04016)*

7.2.1.5 Keep track of active non-default sessions

[SWS_DM_00380]{DRAFT} **Support for S3 timer** ⌈The `Diagnostic Server instance` shall provide support for $S3_{Server}$ (session timeout) with a fixed value of 5 second. The timer handling shall be implemented according to ISO 14229-2[13].⌋*(RS_Diag_04006)*

[SWS_DM_00381]{DRAFT} **Session timeout** ⌈Whenever a non-default session is active and when the session timeout ($S3_{Server}$) is reached without receiving any diagnostic request, the `Diagnostic Server instance` shall reset to the default session state. `Diagnostic Server instance` internal states for service processing shall be reset according to ISO 14229-2[13].⌋*(RS_Diag_04006)*

[SWS_DM_00382]{DRAFT} **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])
- 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])

Start of $S3_{Server}$ means reset the timer and start counting from the beginning.⌋*(RS_Diag_04006)*

[SWS_DM_00383]{DRAFT} **Session timeout stop** ⌈The session timeout timer ($S3_{Server}$) shall be stopped when the reception of an `UDS` message was indicated ([SWS_DM_00309]).⌋*(RS_Diag_04006)*

[SWS_DM_00812]{DRAFT} **Re-enabling on transition to default session** ⌈If DTC setting is disabled and DM is transitioning into default session, then DM shall enable the DTC setting again.⌋*()*

7.2.1.6 UDS service processing

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

[SWS_DM_00127]{DRAFT} **Availability of diagnostic service processors** ⌈The Diagnostic Server instance shall provide a service processor on SID level for all services by existence of a DiagnosticServiceClass referenced by a DiagnosticServiceInstance.serviceClass.⌋*(RS_Diag_04196)*

7.2.1.6.1 Supported UDS Services

The Diagnostic Server instance shall support the following listed UDS services:

SID	Service	Support Type	Reference
0x10	DiagnosticSessionControl	Internally	7.2.1.6.3
0x11	ECUReset	Externally	7.2.1.6.4
0x14	ClearDiagnosticInformation	Internally	7.2.1.6.5
0x19	ReadDTCInformation	Internally	7.2.1.6.6
0x22	ReadDataByIdentifier	Internally & Externally	7.2.1.6.7
0x27	SecurityAccess	Internally & Externally	7.2.1.6.8
0x28	CommunicationControl	Externally	7.2.1.6.9
0x2E	WriteDataByIdentifier	Externally	7.2.1.6.10
0x31	RoutineControl	Externally	7.2.1.6.11
0x34	RequestDownload	Externally	7.2.1.6.12
0x35	RequestUpload	Externally	7.2.1.6.13
0x36	TransferData	Externally	7.2.1.6.14
0x37	RequestTransferExit	Externally	7.2.1.6.15
0x3E	TesterPresent	Internally	7.2.1.6.16
0x85	ControlDTCSetting	Internally	7.2.1.6.17
0x86	ResponseOnEvent	Internally	7.2.1.6.18

Table 7.2: UDS Services supported by Diagnostic Server instance

Note:

- UDS services which are not supported by DM, are documented in the section Known Limitations.
- Support Type Internally means, that the service with the given SID can be completely processed internally within the Diagnostic Server instance without relying on external functionality - typically in form of an AA. Support Type Externally means, that the Diagnostic Server instance needs to call

an external function, to be able to process the service with the given SID. The mixed support Type "Internally & Externally" means, that for the service with the given SID partially calls to an external function have to be done, but it partially could be also handled internally.

7.2.1.6.2 Common service processing items

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

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

[SWS_DM_00129]{DRAFT} **Supported addressAndLengthFormatIdentifier** ⌈The `Diagnostic Server instance` shall support for each nibble of the addressAndLengthFormatIdentifier a value between 1 and 8.⌋*(RS_Diag_04120)*

[SWS_DM_00130]{DRAFT} **Not supported addressAndLengthFormatIdentifier** ⌈The `Diagnostic Server instance` shall send the negative response 0x31 (requestOutOfRange), if an addressAndLengthFormatIdentifier with a value outside the range between 1 and 8 is received.⌋*(RS_Diag_04120)*

7.2.1.6.3 Service 0x10 – DiagnosticSessionControl

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

[SWS_DM_00226]{DRAFT} **Support of UDS service DiagnosticSessionControl** ⌈The `Diagnostic Server instance` shall provide the UDS service 0x10 DiagnosticSessionControl according to ISO 14229-1[1].⌋*(RS_Diag_04198)*

[SWS_DM_00227]{DRAFT} **Check for supported sessions** ⌈If the Subfunction addressed by the DiagnosticSessionControl according to [SWS_DM_00226] is not supported by the configuration, i.e., there is no `DiagnosticSession` configured with `id` matching the requested Subfunction value, the `Diagnostic Server instance` shall return a NRC 0x12 (SubfunctionNotSupported).⌋*(RS_Diag_04196)*

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

[SWS_DM_00228]{DRAFT} **Switch to requested Diagnostic Session** ⌈On positive evaluation of a DiagnosticSessionControl request, the `Diagnostic Server instance` shall send the positive response message. After the response message is sent, the Diagnostic Server shall internally switch to the `DiagnosticSession` with

id matching the requested Subfunction value, and shall set new timing parameters according to the associated parameters p2ServerMax and p2StarServerMax.⌋(*RS_Diag_04198*)

[SWS_DM_00845]{DRAFT} **Notification about session change** ⌈If the Diagnostic Server instance did successfully change the session of a conversation, it shall update the diagnostic session of the according ara::diag::Conversation class ([SWS_DM_00693]) instance internally.⌋(*RS_Diag_04208*)

7.2.1.6.4 Service 0x11 – ECUReset

[SWS_DM_00234]{DRAFT} **Support of UDS service ECUReset** ⌈The Diagnostic Server instance shall provide the UDS service 0x11 ECUReset according to ISO 14229-1[1].⌋(*RS_Diag_04196*)

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

The ExecutionType parameter shall be set:

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

In case the parameter DiagnosticEcuResetClass.respondToReset is present and set to DiagnosticResponseToEcuResetEnum.respondAfterReset to: kDefered⌋(*RS_Diag_04196*)

[SWS_DM_00268]{DRAFT} **EcuReset positive response processing before reset** ⌈If the external processor did NOT raise an ApApplicationError, the Diagnostic Server instance shall return a positive response before the actual reset, in case the parameter DiagnosticEcuResetClass.respondToReset is either not present or present and set to DiagnosticResponseToEcuResetEnum.respondBeforeReset.⌋(*RS_Diag_04019*)

[SWS_DM_00360]{DRAFT} **EcuReset positive response processing after reset** ⌈If the external processor did NOT raise an ApApplicationError, the Diagnostic Server instance shall return a positive response after the actual reset if NotifyReestablishment method (see [SWS_DM_00326]) is called (which could also happen after a restart of DM itself), in case the parameter DiagnosticEcuResetClass.respondToReset is present and set to DiagnosticResponseToEcuResetEnum.respondAfterReset.⌋(*RS_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]{DRAFT} **EcuReset application error processing** ⌈If RequestRestart raised an ApApplicationError contained in RequestRestart, the Diagnostic Server instance shall return a negative response with the value 0x22.⌋*(RS_Diag_04196)*

[SWS_DM_00269]{DRAFT} **Reaction on Unsupported Subfunction** ⌈The Diagnostic Server instance shall send a negative response 0x12 (SubfunctionNotSupported), if the requested subfunction value is neither in configured range of default subfunction values (requestType, see ISO 14229-1[1]) nor in range of the configured DiagnosticEcuReset.customSubFunctionNumber in the ECU.⌋*(RS_Diag_04196)*

7.2.1.6.5 Service 0x14 – ClearDiagnosticInformation

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

[SWS_DM_00090]{DRAFT} **Support of UDS service ClearDiagnosticInformation** ⌈The Diagnostic Server instance shall provide the UDS service 0x14 ClearDiagnosticInformation according to ISO 14229-1[1].⌋*(RS_Diag_04180, RS_Diag_04196)*

[SWS_DM_00091]{DRAFT} **Evaluation of ClearDiagnosticInformation parameters** ⌈The Diagnostic Server instance shall determine the DTC group or single DTC to clear from the 'groupOfDTC' parameter the UDS request.⌋*(RS_Diag_04180, RS_Diag_04117)*

[SWS_DM_00092]{DRAFT} **Parameter range check for groupOfDTC request parameter** ⌈The Diagnostic Server instance 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.⌋*(RS_Diag_04180, RS_Diag_04117)*

[SWS_DM_00113]{DRAFT} **Positive response for UDS service 0x14** ⌈If Diagnostic Server instance has cleared the requested 'groupOfDTC', the Diagnostic Server instance shall send a positive response.⌋*(RS_Diag_04196)*

The DTC clearing behavior is described in detail in section 7.2.2.4.5. It consists of resetting the DTC status and deleting snapshot records and extended data records.

[SWS_DM_00114]{DRAFT} **Limitation to one simultaneous DTC clear operation** ⌈If a DTC clear operation is already in progress, the Diagnostic Server instance shall deny an UDS request 0x14 and send a negative response 0x22 (conditionsNotCorrect).⌋*(RS_Diag_04196)*

[SWS_DM_00115]{DRAFT} **Memory error handling while clearing DTCs** ⌈The Diagnostic Server instance shall return a negative response NRC 0x72 (generalProgrammingFailure) if it encounters an error in the non-volatile memory while clearing the DTCs.⌋*(RS_Diag_04180)*

The definition of a failure of the non-volatile memory is hardware and project specific. In general if the clear DTC operation could not delete the snapshot records, extended data records and if it could not reset the UDS DTC status byte because the underlying storage system reported and error, a non-volatile memory error can be assumed.

[SWS_DM_00122]{DRAFT} **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_00896], the `Diagnostic Server instance` 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_00896]
- it was requested to clear a DTC group and all the DTCs of the DTC group could not be cleared according to [SWS_DM_00896]
  (This doesn't apply when one or more DTC are allowed to be cleared.)

⌋*(RS_Diag_04117)*

[SWS_DM_00159]{DRAFT} **Allow only to clear `GroupOfAllDTCs`** ⌈If the configuration `DiagnosticCommonProps.clearDtcLimitation` is set to `clearAllDtcs`, the `Diagnostic Server instance` shall only allow to clear all DTCs via the `GroupOfAllDTC` as defined in [SWS_DM_00065]. In case a different value is given in groupOfDTC request parameter, the `Diagnostic Server instance` shall return a negative response 0x31 (RequestOutOfRange).⌋*(RS_Diag_04117)*

[SWS_DM_00160]{DRAFT} **Allow to clear single `DTCs`** ⌈If the configuration `DiagnosticCommonProps.clearDtcLimitation` is set to `allSupportedDtcs`, the `Diagnostic Server instance` shall allow to clear single DTCs or DTCGroups. [SWS_DM_00092] defines the possible and refused values.⌋*(RS_Diag_04117)*

[SWS_DM_00162]{DRAFT} **Point in time for positive response for ClearDTC** ⌈The `Diagnostic Server instance` shall send a positive response for a ClearDiagnosticInformation service after all memory is cleared in the server. This is regardless how the `Diagnostic Server instance` memory is organized (splitted, volatile, non-volatile).⌋*(RS_Diag_04180, RS_Diag_04196)*

[SWS_DM_00163]{DRAFT} **Definition of a failed clear operation with event clear allowed and event combination** ⌈If it is requested to clear a single DTC and multiple `DiagnosticEventToTroubleCodeUdsMapping` referencing this `DiagnosticEventToTroubleCodeUdsMapping.troubleCodeUds` the `Diagnostic Server instance` shall send a negative response 0x22 (conditionsNotCorrect) if one event forbids the clearance of the DTC according to [SWS_DM_00896].⌋*(RS_Diag_04180)*

[SWS_DM_00164]{DRAFT} **Definition of a failed clear operation with event clear allowed and clearing a group of DTCs** ⌈If it is requested to clear a group of DTCs,

the Diagnostic Server instance 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_00896].⌋(*RS_Diag_04180*)

7.2.1.6.5.1 Clearing user-defined fault memory

According to [SWS_DM_00090] the Diagnostic Server instance implements an ISO 14229-1[1] compatible UDS service ClearDiagnosticInformation. This implies a limitation that only the primary fault memory can be cleared using this UDS service. To provide means to clear the user-defined fault memories, the Diagnostic Server instance prospectively implements an agreed proposal by ISO 14229-1 to allow clearance of used defined fault memories. The proposal can be found in the ISO 14229 document: "02_ISO_14229-1_Comments-Summary_2016-09-13.docx". Until the next final release of ISO 14229-1[1] containing this extension, the Diagnostic Server instance 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]{DRAFT} **Support of a user-defined fault memory clear request** ⌈If the Diagnostic Server instance receives a a UDS service 0x14 ClearDiagnosticInformation with a length of 5 bytes, the Diagnostic Server instance shall interpret this request as a request to clear user-defined fault memory.⌋(*RS_Diag_04197*)

[SWS_DM_00194]{DRAFT} **Definition of the user-defined fault memory number for ClearDiagnosticInformation** ⌈If the Diagnostic Server instance receives a UDS request to clear user-defined fault memory according to [SWS_DM_00193], the DM shall get the number of user-defined fault memory to be cleared from the fifth byte in the request.⌋(*RS_Diag_04197*)

[SWS_DM_00195]{DRAFT} **Clearing a user-defined memory** ⌈If the Diagnostic Server instance is requested to clear the user-defined fault memory according to [SWS_DM_00193] and an DiagnosticMemoryDestinationUserDefined.memoryId exists with the requested user-defined memory number according to [SWS_DM_00194], the Diagnostic Server instance shall clear the requested user-defined fault memory.⌋(*RS_Diag_04197*)

For details about the fault memory clearing process please also refer to section 7.2.2.4.5.

[SWS_DM_00208]{DRAFT} **Validation of the requested user-defined memory number** ⌈If the Diagnostic Server instance is requested to clear the user-defined fault memory according to [SWS_DM_00193] and no DiagnosticMemoryDestinationUserDefined.memoryId exists with the requested user-defined memory number according to [SWS_DM_00194], the Diagnostic Server instance shall return a NRC 0x31 (RequestOutOfRange).⌋*(RS_Diag_04197)*

7.2.1.6.6 Service 0x19 – ReadDTCInformation

Some UDS responses for the Service "0x19 – ReadDTCInformation" use the parameter "DTCFormatIdentifier" as part of the response PDU. The Diagnostic Server instance 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]{DRAFT} **Mapping between ISO 14229-1[1] and Autosar Diagnostic Extract Template [2] of the DTCFormatIdentifier** ⌈If a positive response for service 0x19 with the ISO 14229-1[1] parameter "DTCFormatIdentifier" is sent, the Diagnostic Server instance shall derive the value from DiagnosticCommonProps.typeOfDtcSupported applying the following mapping rule:⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

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

7.2.1.6.6.1 SF 0x01 – reportNumberOfDTCByStatusMask

[SWS_DM_00244]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x01** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

[SWS_DM_00061]{DRAFT} **Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCByStatusMask** ⌈While sending the positive response for ReadDTCInformation.reportNumberOfDTCByStatusMask, the Diagnostic Server instance shall set the response PDU "DTCFormatIdentifier" according to the mapping of [SWS_DM_00062].⌋*(RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.2 SF 0x02 – reportDTCByStatusMask

[SWS_DM_00245]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x02** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.3 SF 0x04 – reportDTCSnapshotRecordByDTCNumber

[SWS_DM_00246]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x04** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.4 SF 0x06 – reportDTCExtDataRecordByDTCNumber

[SWS_DM_00370]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x06** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.5 SF 0x07 – reportNumberOfDTCBySeverityMaskRecord

[SWS_DM_00247]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x07** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157)*

[SWS_DM_00063]{DRAFT} **Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord**

⌈While sending the positive response for ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord, the Diagnostic Server instance shall set the response PDU "DTCFormatIdentifier" according to the mapping of [SWS_DM_00062].⌋*(RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.6 SF 0x14 – reportDTCFaultDetectionCounter

[SWS_DM_00371]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x14** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.7 SF 0x17 – reportUserDefMemoryDTCByStatusMask

[SWS_DM_00372]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x17** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.8 SF 0x18 – reportUserDefMemoryDTCSnapshotRecordByDTCNumber

[SWS_DM_00373]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x18** ⌈The Diagnostic Server instance 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'.⌋*(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.6.9 SF 0x19 – reportUserDefMemoryDTCExtDataRecordByDTCNumber

[SWS_DM_00374]{DRAFT} **Support of UDS service ReadDTCInformation, Subfunction 0x19** ⌈The Diagnostic Server instance 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'.⌋ *(RS_Diag_04180, RS_Diag_04157, RS_Diag_04067)*

7.2.1.6.7 Service 0x22 – ReadDataByIdentifier

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

[SWS_DM_00170]{DRAFT} **Realisation of UDS service ReadDataByIdentifier (0x22)** ⌈The Diagnostic Server instance shall implement the diagnostic service 0x22 ReadDataByIdentifier according to ISO 14229-1[1].⌋ *(RS_Diag_04196)*

[SWS_DM_00412]{DRAFT} **Check requested number of DataIdentifiers** ⌈On reception of the UDS Service ReadDataByIdentifier (0x22), the Diagnostic Server instance shall check the number of the requested DataIdentifiers against the configuration parameter maxDidToRead.⌋ *(RS_Diag_04203)*

[SWS_DM_00409]{DRAFT} **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.⌋ *(RS_Diag_04203)*

[SWS_DM_00413]{DRAFT} **Check supported DataIdentifier in active session** ⌈On reception of the UDS Service ReadDataByIdentifier (0x22), a requested DataIdentifier shall be considered as supported in active session if and only if the DataIdentifier is supported according to [SWS_DM_00409] and the active session passes the execution permission check as per [SWS_DM_00101].⌋ *(RS_Diag_04203)*

[SWS_DM_00414]{DRAFT} **Check supported DataIdentifier on active security level** ⌈On reception of the UDS Service ReadDataByIdentifier (0x22), a requested DataIdentifier shall be considered as supported on active security level if and only if the DataIdentifier is supported according to [SWS_DM_00409] and the active security level passes the execution permission check as per [SWS_DM_00103].⌋ *(RS_Diag_04203)*

[SWS_DM_00570]{DRAFT} **Retrieving data for requested DataIdentifier** ⌈On reception of the UDS Service ReadDataByIdentifier (0x22), the Diagnostic Server instance shall retrieve the data for a DataIdentifier from the mapped RPortPrototypes.⌋ *(RS_Diag_04097)*

[SWS_DM_00571]{DRAFT} **Reaction on ApplicationError** ⌈If the *Result* of external processor has an error of `ara::diag::DiagUdsNrcErrorDomain`, the `Diagnostic Server instance` shall return a negative response with the value of the error code.⌋*(RS_Diag_04196)*

Note: If multiple DataIdentifer are requested within one ReadDataByIdentifier request, [SWS_DM_00571] might result in a deviation from ISO 14229-1[1] in case the `AA` raises an `ApApplicationError` `kRequestOutOfRange` (resulting in NRC 0x31). According to ISO 14229-1[1], chapter 10.2, a tester expects to receive NRC 0x31 only in case **none** of the requested DataIdentifier are supported. Handling of `ApApplicationErrors` as described in [SWS_DM_00571] might lead to NRC 0x31 on processing one of the requested DataIdentfier without checking the other requested DataIdentifier.

7.2.1.6.8 Service 0x27 – SecurityAccess

[SWS_DM_00236]{DRAFT} **Realization of UDS service 0x27 SecurityAccess** ⌈The `Diagnostic Server instance` shall implement the diagnostic service 0x27 SecurityAccess according to ISO 14229-1[1].⌋*(RS_Diag_04196, RS_Diag_04005)*

[SWS_DM_00863]{DRAFT} **Checking Supported Subfunction for RequestSeed** ⌈On reception of a request for UDS Service SecurityAccess (0x27), the `Diagnostic Server instance` shall call `diag::SecurityAccess::GetSeed` ([SWS_DM_00764]) if the requested subfunction value (access type) matches to the value of the instance of `DiagnosticSecurityAccess` with `requestSeedId`. The `security_access_data_record` parameter of the method `diag::SecurityAccess::GetSeed` ([SWS_DM_00764]) shall be filled with the `securityAccessDataRecord` provided by the tester. If no data is provided by the tester, the `security_access_data_record` parameter shall be empty.⌋*(RS_Diag_04203)*

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

[SWS_DM_00507]{DRAFT} **Length check on UDS Service 0x27 request with Subfunction for RequestSeed** ⌈On reception of a request for UDS Service SecurityAccess (0x27) with subfunction value matching the `requestSeedId` of a configured `DiagnosticSecurityAccess`, the `Diagnostic Server instance` shall perform the message length check against the optionally configured `accessDataRecordSize` of the related `DiagnosticSecurityLevel`. A non-present parameter `accessDataRecordSize` results in a check against 0 additional request bytes. If the length check fails, the `Diagnostic Server instance` shall send NRC 0x13 (IncorrectMessageLengthOrInvalidFormat).⌋*(RS_Diag_04203)*

[SWS_DM_00864]{DRAFT} **Checking Supported Subfunction for CompareKey** ⌈The `Diagnostic Server instance` shall call

`diag::SecurityAccess::CompareKey()` ([SWS_DM_00765]) when the requested subfunction value (access type) - 1 (to get the corresponding requestSeed) is similar to the value of instance of `DiagnosticSecurityAccess` with `requestSeedId`.⌋*(RS_Diag_04203)*

[SWS_DM_00363]{DRAFT} **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).⌋*(RS_Diag_04196)*

[SWS_DM_00846]{DRAFT} **Notification about security-level change** ⌈If `Diagnostic Server instance` did successfully change the security-level of a conversation, it shall update the security level of according `diag::Conversation class` instance internally. Whether a security level is applicable by the `DiagnosticSecurityAccess` is defined by `securityLevel`.⌋*(RS_Diag_04208)*

[SWS_DM_00270]{DRAFT} **Counting of attempts to change security level** ⌈The `Diagnostic Server instance` 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.⌋*(RS_Diag_04208)*

[SWS_DM_00271]{DRAFT} **Evaluate the number of failed security level change attempts** ⌈The `Diagnostic Server instance` shall compare the number of failed `DiagnosticSecurityLevel` changes with threshold value `numFailedSecurityAccess` after each failed attempt.
If the number of failed attempts is below the threshold value `numFailedSecurityAccess` the `Diagnostic Server instance` module shall send a negative response with `NRC` 0x35 (InvalidKey).
If the number of failed attempts reaches the threshold value `numFailedSecurityAccess` the `Diagnostic Server instance` 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.⌋*(RS_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]{DRAFT} **Expiration of the delay timer** ⌈As long as the delay timer (see [SWS_DM_00271]) configured with threshold value `securityDelayTime` has not expired, all requests for `DiagnosticSecurityLevel` change with subfunction value (access type) requestSeed shall be responded with `NRC` 0x37 (requiredTimeDelayNotExpired).
⌋*(RS_Diag_04208)*

[SWS_DM_00478]{DRAFT} **Persistent Storage of failed attempts to change security level** ⌈The Diagnostic Server instance module shall store the number of failed attempts persistently for every security access type separately. (see [SWS_DM_00270])⌋*(RS_Diag_04208)*

[SWS_DM_00479]{DRAFT} **Blocking Timer for security access on Restart or Power down - power up cycle** ⌈The Diagnostic Server instance module shall restart the security delay timer with the higher value of DiagnosticCommonProps.securityDelayTimeOnBoot / DiagnosticSecurityLevel.securityDelayTime of the according DiagnosticSecurityLevel if at least one of the stored numbers of failed attempts are greater or equal than the threshold value DiagnosticSecurityLevel.numFailedSecurityAccess. The behavior is equal to the behavior on runtime [SWS_DM_00272]) In case failed attempts are lower than the threshold value, the handling is equal to the behavior on runtime. (see [SWS_DM_00270] and [SWS_DM_00271])⌋*(RS_Diag_04208)*

[SWS_DM_00480]{DRAFT} **Security Access Blocking Timer** ⌈If DiagnosticSecurityAccessClass.sharedTimer exists and is set to true, a shared delay timer instance and shared value DiagnosticSecurityLevel.securityDelayTime shall be used for all security levels. As long as the blocking timer is running and not expired, all requests for every DiagnosticSecurityLevel change with subfunction value (access type) requestSeed shall be responded with NRC 0x37 (requiredTimeDelayNotExpired). (see [SWS_DM_00272]) If DiagnosticSecurityAccessClass.sharedTimer not exists or is set to false, an independent timer instance and timer value shall be used for each security level.⌋*(RS_Diag_04208)*

[SWS_DM_CONSTR_00208]{DRAFT} **Delay time value for sharedTimer** ⌈If DiagnosticSecurityAccessClass.sharedTimer exists and is set to true, the value DiagnosticSecurityLevel.securityDelayTime shall be identical for all configured security levels.⌋*(RS_Diag_04208)*

7.2.1.6.9 Service 0x28 – CommunicationControl

[SWS_DM_00140]{DRAFT} **Realisation of UDS service 0x28 CommunicationControl** ⌈The Diagnostic Server instance shall implement the diagnostic service 0x28 CommunicationControl according to ISO 14229-1[1].⌋*(RS_Diag_04196)*

[SWS_DM_00252]{DRAFT} **Reaction on Unsupported Subfunction** ⌈The Diagnostic Server instance 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.⌋*(RS_Diag_04203)*

[SWS_DM_00865]{DRAFT} **Communication control service processing** ⌈The Diagnostic Server instance shall call the method

diag::CommunicationControl::CommCtrlRequest ([SWS_DM_00808]) to process a communication control service.⌋(*RS_Diag_04169*)

[SWS_DM_00866]{DRAFT} **Negative Response processing** ⌈If the external processor raised an error according to [SWS_DM_00526], the Diagnostic Server instance shall return a negative response with the value of the error code.⌋(*RS_Diag_04196*)

[SWS_DM_00199]{DRAFT} **Positive Response processing** ⌈If the external processor did raise no ApApplicationError, the Diagnostic Server instance shall return a positive response.⌋(*RS_Diag_04196*)

7.2.1.6.10 Service 0x2E – WriteDataByIdentifier

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

[SWS_DM_00186]{DRAFT} **Realisation of UDS service WriteDataByIdentifier (0x2E)** ⌈The Diagnostic Server instance shall implement the diagnostic service 0x2E WriteDataByIdentifier according to ISO 14229-1[1].⌋(*RS_Diag_04196*)

[SWS_DM_00415]{DRAFT} **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.⌋(*RS_Diag_04203*)

[SWS_DM_00416]{DRAFT} **Check supported DataIdentifier in active session** ⌈On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported in active session if and only if the DataIdentifier is supported according to [SWS_DM_00415] and the active session passes the execution permission check as per [SWS_DM_00101].⌋(*RS_Diag_04203*)

[SWS_DM_00417]{DRAFT} **Check supported DataIdentifier on active security level** ⌈On reception of the UDS Service WriteDataByIdentifier (0x2E), a requested DataIdentifier shall be considered as supported on active security level if and only if the DataIdentifier is supported according to [SWS_DM_00415] and the active security level passes the execution permission check as per [SWS_DM_00103].⌋(*RS_Diag_04203*)

[SWS_DM_00572]{DRAFT} **Writing data for requested DataIdentifier** ⌈On reception of the UDS Service WriteDataByIdentifier (0x2E) the Diagnostic Server instance shall retrieve the data for a DataIdentifier from the mapped RPortPrototypes.⌋(*RS_Diag_04097*)

[SWS_DM_00573]{DRAFT} **Reaction on ApplicationError** ⌈If the *Result* of external processor has an error of `ara::diag::DiagUdsNrcErrorDomain`, the `Diagnostic Server instance` shall return a negative response with the value of the error code.⌋*(RS_Diag_04196)*

7.2.1.6.11 Service 0x31 – RoutineControl

[SWS_DM_00201]{DRAFT} **Realization of UDS service RoutineControl (0x31)** ⌈The `Diagnostic Server instance` shall implement the diagnostic service RoutineControl (0x31) according to ISO 14229-1[1] for subFunctions `startRoutine`, `stopRoutine` and `requestRoutineResults`.⌋*(RS_Diag_04196, RS_Diag_04224)*

[SWS_DM_00202]{DRAFT} **Check for Supported RoutineIdentifier and Reaction** ⌈The `Diagnostic Server instance` shall check, whether the RoutineIdentifier addressed by the UDS Service RoutineControl (0x31) is supported by an existing `DiagnosticRoutine` with a matching `id` in the configuration. If the RoutineIdentifier addressed by the UDS Service RoutineControl (0x31) is not supported a negative response with NRC 0x31 (requestOutOfRange) shall be returned.⌋*(RS_Diag_04203, RS_Diag_04224)*

[SWS_DM_00448]{DRAFT} **Check supported RoutineIdentifier subfunction in active session** ⌈On reception of the UDS Service RoutineControl (0x31), a requested subfunction of a RoutineIdentifiershall be considered as supported in active session if and only if the RoutineIdentifier is supported according to [SWS_DM_00202] and the active session passes the execution permission check. If the session permission check fails, NRC 0x31 shall be returned.⌋*(RS_Diag_04203, RS_Diag_04224)*

[SWS_DM_00437]{DRAFT} **Check supported RoutineIdentifier subfunction on active security level** ⌈On reception of the UDS Service RoutineControl (0x31), a requested subfunction of a RoutineIdentifier shall be considered as supported on active security level if and only if the RoutineIdentifier is supported according to [SWS_DM_00202] and the active security level passes the execution permission check as per [SWS_DM_00450].⌋*(RS_Diag_04203, RS_Diag_04224)*

[SWS_DM_00203]{DRAFT} **Check for Supported Subfunction and Reaction** ⌈The `Diagnostic Server instance` shall check, whether the Subfunction addressed by the UDS Service RoutineControl (0x31) is supported by checking the existence of the corresponding attributes `start` or `stop` or `requestResult` in the related `DiagnosticRoutine` configuration. If the Subfunction addressed by the UDS Service RoutineControl (0x31) is not supported by the configuration a negative response NRC 0x12 (SubfunctionNotSupported) shall be returned.⌋*(RS_Diag_04203, RS_Diag_04224)*

[SWS_DM_00574]{DRAFT} **UDS Service RoutineControl (0x31) startRoutine processing** ⌈The `Diagnostic Server instance` shall call the `diag::GenericRoutine::Start` ([SWS_DM_00554]) or `Routine::Start`

([SWS_DM_00591]) according to the mapped diagnostic interface to process the subfunction `startRoutine`.⌋*(RS_Diag_04196, RS_Diag_04224)*

[SWS_DM_00575]{DRAFT} **UDS Service RoutineControl (0x31) requestRoutineResults processing** ⌈The `Diagnostic Server instance` shall call `diag::GenericRoutine::RequestResults()` ([SWS_DM_00554]) or `Routine::RequestResults` ([SWS_DM_00593]) according to the mapped diagnostic interface to process the subfunction `requestRoutineResults`.⌋*(RS_Diag_04196, RS_Diag_04224)*

[SWS_DM_00576]{DRAFT} **UDS Service RoutineControl (0x31) stopRoutine processing** ⌈The `Diagnostic Server instance` shall call `Routine::Stop` ([SWS_DM_00592]) or `diag::GenericRoutine::Stop` ([SWS_DM_00555]) according to the mapped diagnostic interface to process the subfunction `stopRoutine`.⌋ *(RS_Diag_04196, RS_Diag_04224)*

7.2.1.6.12 Service 0x34 – RequestDownload

[SWS_DM_00128]{DRAFT} **Realization of UDS service RequestDownload (0x34)** ⌈The `Diagnostic Server instance` shall implement the UDS service RequestDownload (0x34) according to ISO 14229-1[1].⌋*(RS_Diag_04196, RS_Diag_04033)*

[SWS_DM_00446]{DRAFT} **Check Support of UDS service RequestDownload (0x34) in active session** ⌈On reception of the UDS service RequestDownload (0x34), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101].⌋*(RS_Diag_04203)*

[SWS_DM_00447]{DRAFT} **Check Support of UDS service RequestDownload (0x34) on active security level** ⌈On reception of the UDS service RequestDownload (0x34), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103].⌋*(RS_Diag_04203)*

[SWS_DM_00867]{DRAFT} **UDS service RequestDownload (0x34) processing** ⌈The `Diagnostic Server instance` shall call `diag::DownloadService::RequestDownload` ([SWS_DM_00789]) to process an UDS service RequestDownload (0x34).⌋*(RS_Diag_04196)*

7.2.1.6.13 Service 0x35 – RequestUpload

[SWS_DM_00134]{DRAFT} **Realization of UDS service RequestUpload (0x35)** ⌈The `Diagnostic Server instance` shall implement the UDS service RequestUpload (0x35) according to ISO 14229-1[1].⌋*(RS_Diag_04196)*

[SWS_DM_00438]{DRAFT} **Check Support of UDS service RequestUpload (0x35) in active session** ⌈On reception of the UDS service RequestUpload (0x35), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101].⌋*(RS_Diag_04203)*

[SWS_DM_00439]{DRAFT} **Check Support of UDS service RequestUpload (0x35) on active security level** ⌈On reception of the UDS service RequestUpload (0x35), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103].⌋*(RS_Diag_04203)*

[SWS_DM_00868]{DRAFT} **UDS service RequestUpload (0x35) processing** ⌈The `Diagnostic Server instance` shall call `diag::UploadService::RequestUpload` ([SWS_DM_00799]) to process a UDS service RequestUpload (0x35).⌋*(RS_Diag_04033)*

7.2.1.6.14 Service 0x36 – TransferData

[SWS_DM_00137]{DRAFT} **Realization of UDS service TransferData (0x36)** ⌈The `Diagnostic Server instance` shall implement the UDS service TransferData (0x36) according to ISO 14229-1[1].⌋*(RS_Diag_04196)*

[SWS_DM_00440]{DRAFT} **Check Support of UDS service TransferData (0x36) in active session** ⌈On reception of the UDS service TransferData (0x36), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101].⌋*(RS_Diag_04203)*

[SWS_DM_00441]{DRAFT} **Check Support of UDS service TransferData (0x36) on active security level** ⌈On reception of the UDS service TransferData (0x36), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103].⌋*(RS_Diag_04203)*

[SWS_DM_00869]{DRAFT} **UDS service TransferData (0x36) processing** ⌈The `Diagnostic Server instance` shall call `diag::GenericUDSService::HandleMessage()` ([SWS_DM_00618]) to process an UDS service TransferData (0x36).⌋*(RS_Diag_04033)*

ISO 14229-1[1] provides a UDS service TransferData (0x36) specific NRC evaluation sequence. This sequence has checks that in rotating order needs to be done by the `Diagnostic Server instance` 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 class" has only one single method this is not possible for the "GenericUDSService class". As a result of this, the entire service specific NRC handling is inside the "GenericUDSService class" for UDS service TransferData (0x36).

[SWS_DM_00870]{DRAFT} **UDS service TransferData (0x36) validation** ⌈The Diagnostic Server instance shall realize all service specific NRC validation with diag::GenericUDSService [SWS_DM_00602].⌋*(RS_Diag_04033)*

7.2.1.6.15 Service 0x37 – RequestTransferExit

[SWS_DM_00141]{DRAFT} **Realization of UDS service RequestTransferExit (0x37)** ⌈The Diagnostic Server instance shall implement the UDS service RequestTransferExit (0x37) according to ISO 14229-1[1].⌋*(RS_Diag_04196)*

[SWS_DM_00442]{DRAFT} **Check Support of UDS service RequestTransferExit (0x37) in active session** ⌈On reception of the UDS service RequestTransferExit (0x37), the service shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101].⌋ *(RS_Diag_04203)*

[SWS_DM_00443]{DRAFT} **Check Support of UDS service RequestTransferExit (0x37) on active security level** ⌈On reception of the UDS service RequestTransferExit (0x37), the service shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103].⌋*(RS_Diag_04203)*

[SWS_DM_00871]{DRAFT} **UDS service RequestTransferExit (0x37) processing** ⌈The Diagnostic Server instance shall call diag::GenericUDSService::HandleMessage ([SWS_DM_00618]) to process a UDS service RequestTransferExit (0x37).⌋*(RS_Diag_04033)*

[SWS_DM_00872]{DRAFT} **UDS service RequestTransferExit (0x37) validation** ⌈The Diagnostic Server instance shall realize all service specific NRC validation with the diag::GenericUDSService class ([SWS_DM_00602]) of the service processors.⌋*(RS_Diag_04033)*

7.2.1.6.16 Service 0x3E – TesterPresent

[SWS_DM_00126]{DRAFT} **Realisation of UDS service 0x3E TesterPresent** ⌈The Diagnostic Server instance shall internally implement the diagnostic service 0x3E TesterPresent according to ISO 14229-1[1].⌋*(RS_Diag_04196)*

7.2.1.6.17 Service 0x85 – ControlDTCSetting

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

[SWS_DM_00229]{DRAFT} **Support of UDS service ControlDTCSetting (0x85)** ⌈The Diagnostic Server instance shall provide the UDS service ControlDTCSetting (0x85) according to ISO 14229-1[1].⌋*(RS_Diag_04180, RS_Diag_04159)*

[SWS_DM_00444]{DRAFT} **Check Support of UDS service ControlDTCSetting (0x85) in active session** ⌈On reception of the UDS service ControlDTCSetting (0x85), a requested subfunction shall be considered as supported in active session if and only if the active session passes the execution permission check as per [SWS_DM_00101].⌋*(RS_Diag_04203)*

[SWS_DM_00445]{DRAFT} **Check Support of UDS service ControlDTCSetting (0x85) on active security level** ⌈On reception of the UDS service ControlDTCSetting (0x85), a requested subfunction shall be considered as supported on active security level if and only if the active security level passes the execution permission check as per [SWS_DM_00103].⌋*(RS_Diag_04203)*

[SWS_DM_00230]{DRAFT} **Check for supported subfunctions** ⌈If the Subfunction addressed by the UDS service ControlDTCSetting (0x85) according to [SWS_DM_00229] is not supported by the configuration, i.e., there is no DiagnosticControlDTCSetting configured with dtcSettingParameter matching the requested Subfunction value, the Diagnostic Server instance shall return a NRC 0x12 (SubfunctionNotSupported).⌋*(RS_Diag_04203)*

[SWS_DM_00231]{DRAFT} **Invalid value for optional request parameter** ⌈If the Diagnostic Server instance receives a UDS service ControlDTCSetting (0x85) request with DTCSettingControlOptionRecord != 0xFFFFFF, the Diagnostic Server instance shall send a NRC 0x31 (RequestOutOfRange).⌋*(RS_Diag_04203, RS_Diag_04115)*

[SWS_DM_00909]{DRAFT} **Support of Subfunction 0x01 (ON)** ⌈If the Diagnostic Server instance receives a valid UDS service ControlDTCSetting (0x85) with Subfunction 0x01 (ON) and optionally with DTCSettingControlOptionRecord of value 0xFFFFFF, the Diagnostic Server instance shall:

- enable the update of the UDS DTC status byte
- enable the storage in event memory
- update `diag::DTCInformation::ControlDtcStatusType` ([SWS_DM_00663]) to `kDTCSettingOn`

⌋*(RS_Diag_04180, RS_Diag_04159)*

[SWS_DM_00910]{DRAFT} **Support of Subfunction 0x02 (OFF)** ⌈If the Diagnostic Server instance receives a valid UDS service ControlDTCSetting (0x85) with Subfunction 0x02 (OFF) and optionally with DTCSettingControlOptionRecord of value 0xFFFFFF, the Diagnostic Server instance shall:

- disable the update of the UDS DTC status byte
- disable the storage in event memory

- update `diag::DTCInformation::ControlDtcStatusType` ([SWS_DM_00663]) to `kDTCSettingOff`

⌋(*RS_Diag_04180, RS_Diag_04159*)

[SWS_DM_00811]{DRAFT} **Re-enabling of ControlDTCSetting by Diagnostic Application** ⌈In case the DTCSetting is disabled and the Diagnostic Server receives a call EnableControlDtc function ([SWS_DM_00674]) the `Diagnostic Server instance` shall:

- enable the update of the `UDS DTC status byte`
- enable the storage in event memory
- update `diag::DTCInformation::ControlDtcStatusType` ([SWS_DM_00663]) to `kDTCSettingOn`

⌋()

Hint: The monitoring application is responsible for the re-enabling of ControlDTCSetting in case some conditions or states demands so. For this purpose the application can use the interface *diag::DTCInformation* with the function EnableControlDtc() ([SWS_DM_00674]).

7.2.1.6.18 Service 0x86 – ResponseOnEvent

With the UDS Service ResponseOnEvent (0x86), a tester requests an ECU to start or stop transmission of responses initiated by a specified event. Upon registering an event for transmission, the tester also specifies the corresponding service to respond to (e.g: UDS Service ReadDataByIdentifier 0x22).

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

Table 7.3: Supported sub function of ResponseonEvent (0x86)

[SWS_DM_00491]{DRAFT} **Realisation of UDS service 0x86 ResponseOnEvent** ⌈The DM shall internally implement the diagnostic service 0x86 ResponseOnEvent according to ISO 14229-1[1].⌋(*RS_Diag_04160*)

[SWS_DM_00492]{DRAFT} **Client Server communication** ⌈The service ResponseOnEvent is related to a distinct client, i.e. the client performing the ResponseOnEvent initialisation receives the serviceToRespondTo-responses.⌋*(RS_Diag_04160)*

[SWS_DM_00493]{DRAFT} **Reestablishing of Client Server communication** ⌈In case of a canceled diagnostic conversation this client receives the serviceToRespondTo-responses after a successful reestablishing of a diagnostic conversation.⌋*(RS_Diag_04160)*

[SWS_DM_00494]{DRAFT} **Supported sub functions of ResponseOnEvent service** ⌈The client can request different subfunctions of service ResponseOnEvent to initialised ResponseOnEvent services. The ECU supported subfunctions are listed in Table 7.3 Supported sub function of Response on Event (0x86).⌋*(RS_Diag_04160)*

[SWS_DM_00495]{DRAFT} **Start initialisation of ResponseOnEvent** ⌈The subfunction startResponseOnEvent shall always control all initialised ResponseOnEvent services.⌋*(RS_Diag_04160)*

[SWS_DM_00496]{DRAFT} **Stop initialisation of ResponseOnEvent** ⌈The subfunction stopResponseOnEvent shall always control all initialised ResponseOnEvent services.⌋*(RS_Diag_04160)*

[SWS_DM_00497]{DRAFT} **Clear initialisation of ResponseOnEvent** ⌈The subfunction clearResponseOnEvent shall set the ResponseOnEvent services to status ResponseOnEvent-cleared.⌋*(RS_Diag_04160)*

[SWS_DM_00498]{DRAFT} **Exclusive ResponseOnEvent ressources** ⌈There is only one ResponseOnEvent resource per server which can be used by multiple clients.⌋*(RS_Diag_04160)*

[SWS_DM_00499]{DRAFT} **Replacement of a not started ResponseOnEvent initialisation** ⌈If a new ResponseOnEvent initialisation is requested from a second client while a previous ResponseOnEvent initialisation is not started the new ResponseOnEvent initialisation replaces the previous ResponseOnEvent initialisation.⌋*(RS_Diag_04160)*

[SWS_DM_00500]{DRAFT} **Replacement of a started ResponseOnEvent initialisation** ⌈If a new ResponseOnEvent initialisation is requested from a second client while a previous ResponseOnEvent initialisation is active the ResponseOnEvent services have to be stopped and the new ResponseOnEvent initialisation replaces the previous ResponseOnEvent initialisation.⌋*(RS_Diag_04160)*

[SWS_DM_00501]{DRAFT} **Behavior while trying ResponseOnEvent activation while ResponseOnEvent is not initialised** ⌈A NRC 0x24 has to be sent if a ResponseOnEvent service is not initialised.⌋*(RS_Diag_04160)*

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

7.2.1.6.19 Custom Diagnostic Services

[SWS_DM_00502]{DRAFT} **Support for Custom Diagnostic Services** ⌈Custom Diagnostic Services shall be supported according to ISO 14429-1[1], table 2 - service identifier values, to allow UDS services, which are defined by OEM / system suppliers.⌋*(RS_Diag_04177)*

Meta-class `DiagnosticCustomServiceInstance` can be used to define the instance of a Custom Service. Modeling of Custom Diagnostic Services is described in the TPS Manifest Specification [[12]].

7.2.1.7 Cancellation of a Diagnostic Conversation

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

- Replacement by a newly requested Diagnostic Conversation according to [SWS_DM_00431],
- Maximum number of busy responses reached (according to [SWS_DM_00369])

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

[SWS_DM_00482]{DRAFT} **Cancellation of a Diagnostic Conversation** ⌈Cancellation of a `Diagnostic Conversation` shall be performed according to [SWS_DM_00277], [SWS_DM_00278], [SWS_DM_00279], [SWS_DM_00280], [SWS_DM_00847].⌋*(RS_Diag_04167)*

[SWS_DM_00277]{DRAFT} **Cancellation of a Diagnostic Conversation in case of External Service Processing** ⌈On Cancellation of a `Diagnostic Conversation` in case a diagnostic request is currently processed on this `Diagnostic Conversation` by a service processor external to the `Diagnostic Server instance`, the `Diagnostic Server instance` shall notify this external service processor, that the processing for this service shall be canceled according to [SWS_DM_00577].⌋*(RS_Diag_04167)*

[SWS_DM_00278]{DRAFT} **Cancellation of a Diagnostic Conversation in case of Internal Processing** ⌈On Cancellation of a `Diagnostic Conversation` in case a diagnostic request is currently processed on this protocol internally within the `Diagnostic Server instance`, the `Diagnostic Server instance` shall abort started activity as far as possible.⌋*(RS_Diag_04167)*

[SWS_DM_00279]{DRAFT} **Cancellation of a Diagnostic Conversation before Response Transmission** ⌈On Cancellation of a `Diagnostic Conversation` in case a diagnostic request is currently processed on this protocol and response transmission has not yet been started, the `Diagnostic Server instance` shall skip sending any response, which implies **not** to call `Transmit` ([SWS_DM_00327]) of the respective UDS Transport Protocol Handler.⌋*(RS_Diag_04167)*

[SWS_DM_00280]{DRAFT} **Cancellation of a Diagnostic Conversation at Response Transmission** ⌈On Cancellation of a Diagnostic Conversation in case a diagnostic request is currently processed on this Diagnostic Conversation and Transmit ([SWS_DM_00327]) of the UDS TransportLayer was already called, nothing has to be done by the Diagnostic Server instance. This implies a sent out response.⌋ *(RS_Diag_04167)*

[SWS_DM_00847]{DRAFT} **Reinitialization of Service Instance on Cancellation of a Diagnostic Conversation** ⌈On Cancellation of a Diagnostic Conversation, the Diagnostic Server instance shall reset the values of the fields of the associated diag::Conversation class Instance according to [SWS_DM_00843].⌋ *(RS_Diag_04167)*

[SWS_DM_00577]{DRAFT} **Canceling external service processors** ⌈External service processors, which are supporting a CancellationHandler shall be signaled via the ara::diag::CancellationHandler ([SWS_DM_00608]) to cancel their processing.⌋ *(RS_Diag_04167)*

7.2.2 Diagnostic Event Management

7.2.2.1 Diagnostic Events

7.2.2.1.1 Definition

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

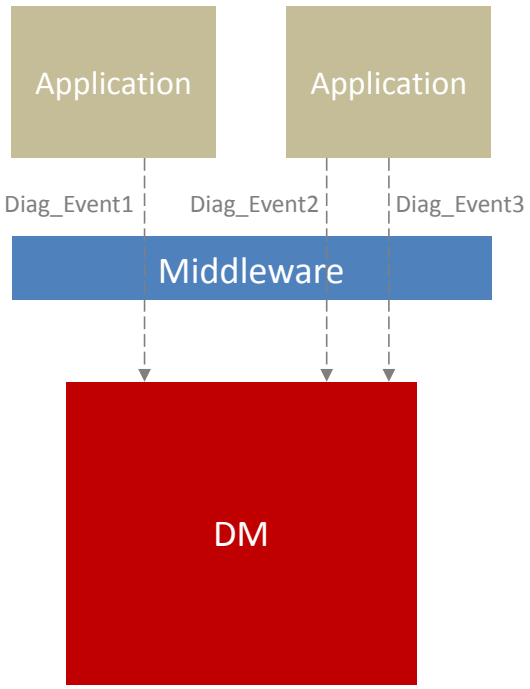

Figure 7.3: Example of diagnostic event usage

[SWS_DM_00007]{DRAFT} **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. This implies that only one application can report a certain `event` and the event reporting interface is explicitly not re-entrant.⌋*(RS_Diag_04063, RS_Diag_04179)*

[SWS_DM_00873]{DRAFT} **Diagnostic event processing interface** ⌈The DM shall provide an instance of `ara::diag::Event` ([SWS_DM_00646]) per configured `DiagnosticEvent`.⌋*(RS_Diag_04179)*

The available `events` are derived from `DiagnosticEvent`.

[SWS_DM_00165]{DRAFT} **Considering only events referencing a `DTC`** ⌈The DM shall consider configured events according to [SWS_DM_00873] only if a `DiagnosticEventToTroubleCodeUdsMapping` exists referencing the `DiagnosticEvent` and a `DiagnosticEventToTroubleCodeUdsMapping.troubleCodeUds`.⌋*(SRS_Diag_04180)*

7.2.2.1.2 Monitors

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

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

Besides to the reporting direction of the monitors (AAs report the monitoring status towards the DM), there is also a connection in the opposite direction: The DM uses the initMonitor notifier method of the ara::diag::Monitor::Monitor ([SWS_DM_00548], [SWS_DM_00549] or [SWS_DM_00550]) instance to trigger a (re-)initialization of diagnostic monitors in the AA. The initMonitor notifier method is registered via the ara::diag::Monitor::Monitor constructors.

[SWS_DM_00562]{DRAFT} **Monitor initialization for clearing reason** ⌈If an associated DTC, belonging to the current monitoring path, was actually cleared via the Clear() function of the ara::diag::DTCInformation instance ([SWS_DM_00671]), the DM shall call the registered initMonitor notifier method with the parameter InitMonitorReason ([SWS_DM_00548], [SWS_DM_00549] or [SWS_DM_00550]) set to kClear ([SWS_DM_00540]).⌋*(RS_Diag_04185)*

[SWS_DM_00563]{DRAFT} **Monitor initialization for operation cycle restart reason** ⌈If a diagnostic event was (re)started by calling the ara::diag::OperationCycle method SetOperationCycle ([SWS_DM_00756]) with the parameter kOperationCycleStart ([SWS_DM_00750]), the DM shall call the registered initMonitor notifier method with the parameter InitMonitorReason ([SWS_DM_00548], [SWS_DM_00549] or [SWS_DM_00550]) set to value kRestart ([SWS_DM_00540]).⌋*(RS_Diag_04186)*

[SWS_DM_00564]{DRAFT} **Monitor initialization for enable condition re-enabling reason** ⌈In case an enable condition mapped to the diagnostic event was changed to fulfilled and in this way all related enable conditions of the event were fulfilled, the DM shall call the registered initMonitor() notifier method ([SWS_DM_00548]) with initMonitorReason parameter set to the value kReenabled ([SWS_DM_00540]).⌋*(RS_Diag_04125, RS_Diag_04192)*

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

[SWS_DM_00565]{DRAFT} **Monitor initialization for DTC setting re-enabling reason** ⌈In case DTC-setting is re-enabled via the UDS service request ControlDTCSetting - 0x85 (see ISO 14229-1[1]), the DM shall call the registered initMonitor() notifier method ([SWS_DM_00548]) with initMonitorReason parameter set to the value kReenabled ([SWS_DM_00540]).⌋*(RS_Diag_04125, RS_Diag_04159)*

For reference see paragraph 7.2.1.6.17.

7.2.2.1.3 Reporting

Per diagnostic monitor an instance of the class ara::diag::Monitor ([SWS_DM_00542]) is created by the application. Diagnostic results are reported to the DM via the method ReportMonitorAction() ([SWS_DM_00543]) of class ara::diag::Monitor. The method ReportMonitorAction() calculates the update of the corresponding instance of ara::diag::Event ([SWS_DM_00646]) (from DiagnosticEvent) and the UDS DTC status byte as well as the storage in the event memory and the capturing of DTC related data. The DM provides also means to ignore a certain call of ReportMonitorAction() in some situations.

[SWS_DM_00567]{DRAFT} **Ignoring reported events for not started operation cycles** ⌈If the function ReportMonitorAction() ([SWS_DM_00543]) was called and the referenced DiagnosticOperationCycle of this reported DiagnosticEvent (via DiagnosticEventToOperationCycleMapping) is set to kOperationCycleEnd, the DM shall do no processing and set the error kReportIgnored to the Result.⌋*(RS_Diag_04178)*

For more details about operation cycles see subsubsection 7.2.2.3.

7.2.2.1.4 Debouncing

Debouncing of reported events is the capability of the DM to filter out undesirable noise reported by monitors. This is used to mature the result of the monitor.

Debouncing means that a report from a monitor does not immediately lead to a change of the UDS DTC status bit kTestFailed but that a delaying threshold value must be reached before. This results in the states for ara::diag::Event::DebouncingState (compare [SWS_DM_00645]). If this threshold value is reached (FDC-equivalent is +127 (FDC_{max}) or -128 (FDC_{min})), the DebouncingState is either kFinallyDefective or kFinallyHealed. This finally also leads to a change of the UDS DTC status bit kTestFailed.

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

- Counter-based debouncing
- Time-based debouncing

Besides the here described debouncing algorithms within the DM implementation, there is also the possibility to do the debouncing monitor-internal within the AA (compare [SWS_DM_00548]). But since this is not part of the DM, no further details are given here.

Which algorithm is used can be configured on a per event basis.

[SWS_DM_00013]{DRAFT} **Events without debouncing** ⌈If an event is not referenced by any DiagnosticEventToDebounceAlgorithmMapping.diagnosticEvent, the DM shall not use a debounce algorithm for this event.⌋*(RS_Diag_04068)*

A monitoring application will call the ReportMonitorAction() ([SWS_DM_00543]) with kPrepassed or kPrefailed ([SWS_DM_00541]) for events, that are debounced by the DM.

[SWS_DM_00874]{DRAFT} **Reporting kPrepassed or kPrefailed for events without an assigned debouncing algorithm** ⌈A new received ReportMonitorAction ([SWS_DM_00543]) with kPrepassed or kPrefailed ([SWS_DM_00541]) for an diagnostic event without assigned debouncing algorithm, the DM shall interpret a reported kPrepassed as kPassed and kPrefailed as kFailed.⌋*(RS_Diag_04068)*

7.2.2.1.4.1 Counter-based debouncing

Counter-based debouncing is done on a per event based counting policy of reported kPrepassed or kPrefailed ([SWS_DM_00541]) 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]{DRAFT} **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.⌋*(RS_Diag_04068)*

[SWS_DM_00018]{DRAFT} **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 start-up. 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.⌋*(RS_Diag_04124)*

[SWS_DM_00028]{DRAFT} **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.⌋*(RS_Diag_04124)*

[SWS_DM_00017]{DRAFT} **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.⌋*(RS_Diag_04125, RS_Diag_04190)*

[SWS_DM_00875]{DRAFT} **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 related monitor calls the method ReportMonitorAction ([SWS_DM_00543]) with kPrefailed ([SWS_DM_00541]).⌋*(RS_Diag_04125, RS_Diag_04068)*

[SWS_DM_00024]{DRAFT} **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 kFinallyDefective ([SWS_DM_00645]).⌋*(RS_Diag_04125, RS_Diag_04068)*

[SWS_DM_00876]{DRAFT} **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 related monitor calls the method ReportMonitorAction ([SWS_DM_00543]) with kPrepassed ([SWS_DM_00541]).⌋*(RS_Diag_04125, RS_Diag_04068)*

[SWS_DM_00025]{DRAFT} **Qualified passed event using counter-based debouncing** ⌈If the internal debounce counter is less or equal to DiagEventDebounceCounterBased.counterPassedThreshold the DM shall process the event as kFinallyHealed ([SWS_DM_00645]).⌋*(RS_Diag_04125, RS_Diag_04068)*

[SWS_DM_00021]{DRAFT} **Direct failed qualification of counter-based events** ⌈If the monitor reports kFailed, the DM shall set the internal debounce counter to the value DiagEventDebounceCounterBased.counterFailedThreshold and process the event as kFinallyDefective ([SWS_DM_00645]).⌋*(RS_Diag_04125, RS_Diag_04068)*

[SWS_DM_00029]{DRAFT} **Direct passed qualification of counter-based events** ⌈If the monitor reports kPassed, the DM shall set the internal debounce counter to the value DiagEventDebounceCounterBased.counterPassedThreshold and process the event as kFinallyHealed ([SWS_DM_00645]).⌋*(RS_Diag_04125, RS_Diag_04068)*

[SWS_DM_00022]{DRAFT} **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 kPrefailed 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_00875] shall be done.⌋*(RS_Diag_04068)*

[SWS_DM_00023]{DRAFT} **Debounce counter jump down behavior** ⌈If kPrepassed is reported for an event and the current debounce counter value is greater than DiagEventDebounceCounterBased.counterJumpDownValue and counterJumpDown is set to true for this 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_00876] shall be done.⌋(*RS_Diag_04068*)

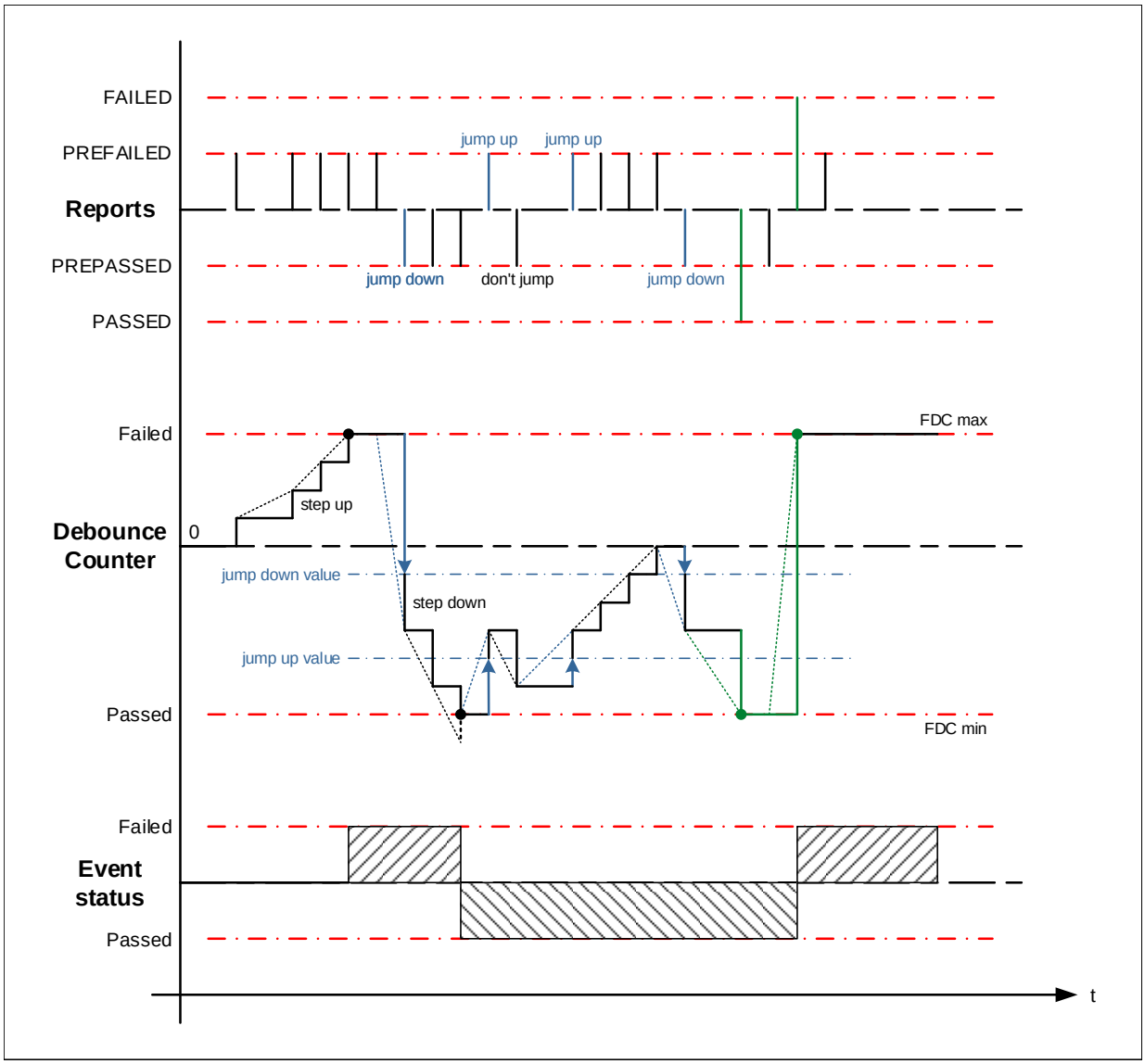

Figure 7.4: Counter-based debouncing

7.2.2.1.4.2 Time-based debouncing

Time-based debouncing is done on a per event based counting policy of reported kPrepassed or kPrefailed 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]{DRAFT} **Use of timer based debouncing for events** ⌈The existence of a DiagnosticEventToDebounceAlgorithmMapping with an aggregation of DiagEventDebounceTimeBased by the referenced DiagnosticDebounceAlgorithmProps.debounceAlgorithm shall activate a time-based debouncing for this event.⌋*(RS_Diag_04225)*

[SWS_DM_00085]{DRAFT} **Internal debounce counter init** ⌈The DM shall initialize the event's internal debounce counter to '0' upon start-up.⌋*(RS_Diag_04225)*

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

[SWS_DM_00030]{DRAFT} **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.⌋*(RS_Diag_04225, RS_Diag_04190)*

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

[SWS_DM_00877]{DRAFT} **Starting time-based event debouncing for failed** ⌈The DM module shall start the debounce timer when the related monitor calls the method ReportMonitorAction ([SWS_DM_00543]) with kPrefailed ([SWS_DM_00541]) to qualify the reported event as kFinallyDefective only when the following conditions are met:

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

⌋*(RS_Diag_04225)*

[SWS_DM_00032]{OBSOLETE} **Restrictions on restarting a running event debounce timer for failed** ⌈Obsolete, since redundant to [SWS_DM_00877]. If the debounce timer of a specific event is already counting towards kFinallyDefective, the DM shall not restart the debounce timer upon a further report of kPrefailed.⌋ *(RS_Diag_04225)*

[SWS_DM_00033]{DRAFT} **Debounce timer behavior upon reported failed** ⌈If the monitor reports kFailed, the DM shall set the debounce timer value to DiagEventDebounceTimeBased.timeFailedThreshold and process the event as kFinallyDefective.⌋*(RS_Diag_04225)*

[SWS_DM_00878]{DRAFT} **Starting time-based event debouncing for passed** ⌈The DM module shall start the debounce timer when the related monitor calls the method ReportMonitorAction ([SWS_DM_00543]) with kPrepassed ([SWS_DM_00541]) to qualify the reported event as kFinallyHealed only when the following conditions are met:

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

⌋(*RS_Diag_04225*)

[SWS_DM_00035]{OBSOLETE} **Restrictions on restarting a running event debounce timer for passed** ⌈Obsolete, since redundant to [SWS_DM_00878]. If the debounce timer of a specific event is already counting towards `kFinallyHealed`, the `DM` shall not restart the debounce timer upon a further report of `kPrepassed`.⌋(*RS_Diag_04225*)

[SWS_DM_00036]{DRAFT} **Debounce timer behavior upon reported passed** ⌈If the `monitor` reports `kPassed`, the `DM` shall set the debounce timer value to `DiagEventDebounceTimeBased.timePassedThreshold` and process the event as `kFinallyHealed`.⌋(*RS_Diag_04225*)

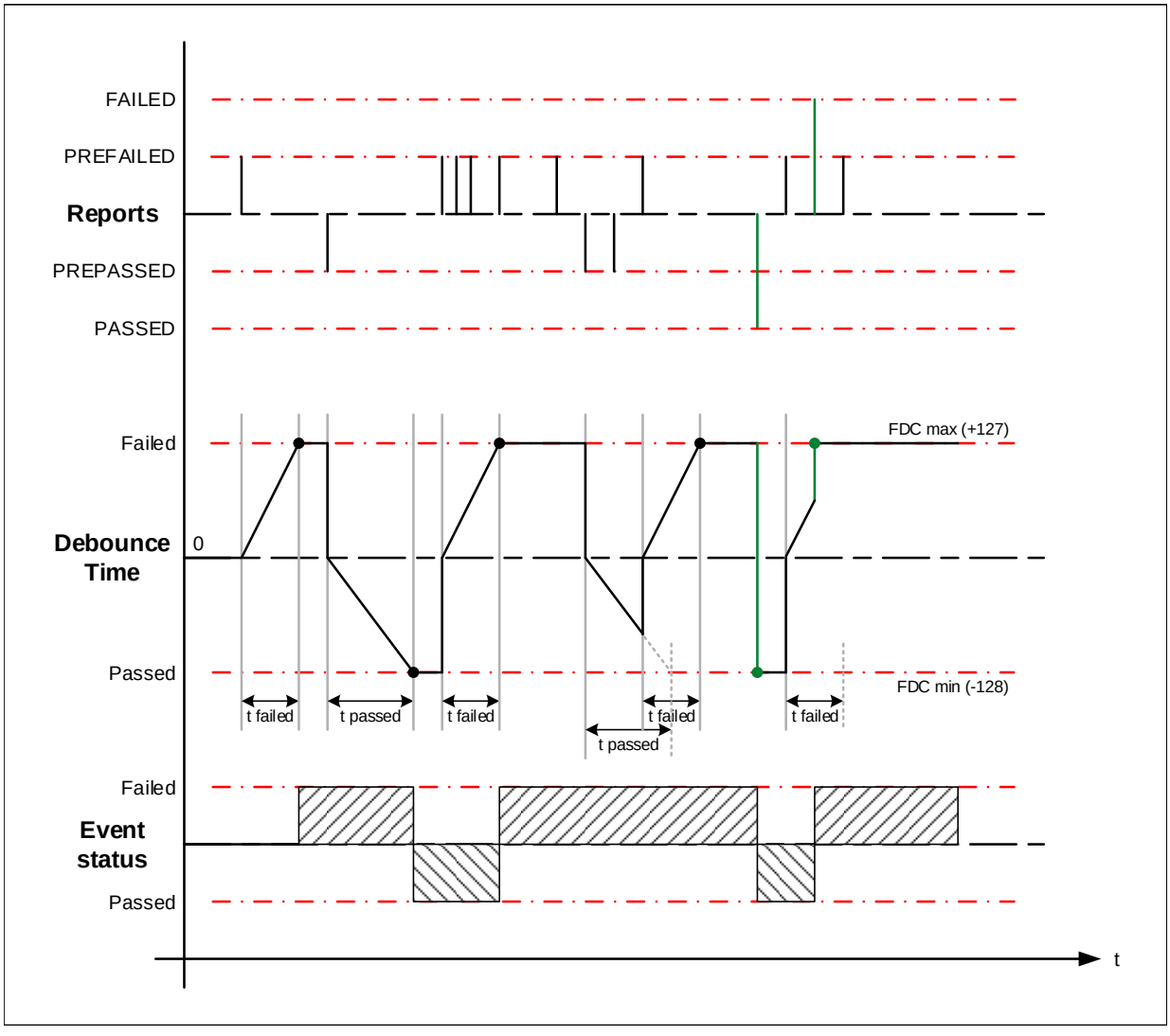

Figure 7.5: Timer based debouncing

[SWS_DM_00880]{DRAFT} **Debounce time freeze request** ⌈If the `ReportMonitorAction` ([SWS_DM_00543]) method of a `ara::diag::Monitor` instance is called with `kFreezeDebouncing` ([SWS_DM_00541]), the `DM` shall freeze the related debounce timer of the corresponding `event`.⌋(*RS_Diag_04068, RS_Diag_04225*)

Freezing of the timer is only supported for events with DiagEventDebounceTimeBased debouncing.

[SWS_DM_00038]{DRAFT} **Continuing a frozen debounce timer** ⌈If a debounce timer is frozen (i.e. the corresponding monitor has called ReportMonitorAction() with kFreezeDebouncing (see [SWS_DM_00541])) and a new kPrepassed or kPrefailed is reported for this event, the DM module shall continue running the debounce timer starting with the frozen value.⌋*(RS_Diag_04225)*

7.2.2.1.4.3 Debounce algorithm reset

In some situations the application might want to reset the debouncing or to freeze it. The DM provides the parameters kFreezeDebouncing and kResetDebouncing ([SWS_DM_00541]) for the method ReportMonitorAction ([SWS_DM_00543]) of class ara::diag::Monitor to provide some means of external control of the debounce counter.

[SWS_DM_00040]{DRAFT} **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.⌋*(RS_Diag_04068, RS_Diag_04225)*

Only on the next call of ReportMonitorAction() with kPrepassed or with kPrefailed the debouncing shall start over again.

[SWS_DM_00879]{DRAFT} **Application resetting the debounce counter** ⌈If the ReportMonitorAction ([SWS_DM_00543]) method of a ara::diag::Monitor instance is called with kResetDebouncing ([SWS_DM_00541]), the DM shall reset the debounce counter or timer of the corresponding event.⌋*(RS_Diag_04068, RS_Diag_04225)*

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

[SWS_DM_00039]{DRAFT} **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.⌋*(RS_Diag_04068, RS_Diag_04225)*

[SWS_DM_00086]{DRAFT} **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.⌋*(RS_Diag_04068, RS_Diag_04225)*

7.2.2.1.4.4 Dependencies to enable conditions

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

[SWS_DM_00881]{DRAFT} **Enable condition influence on debouncing behavior (freeze)** ⌈If the enable conditions are not fulfilled for an event according to [SWS_DM_00568] and the debounce algorithm referenced by that event has the DiagnosticDebounceAlgorithmProps.debounceBehavior set to freeze, the DM shall freeze the according debounce timer or counter for the time the enable conditions are not fulfilled. This means that the debounce timer or counter remains unchanged.⌋*(RS_Diag_04192, RS_Diag_04125)*

[SWS_DM_00882]{DRAFT} **Enable condition influence on debouncing behavior (reset)** ⌈If the enable conditions are not fulfilled for an event according to [SWS_DM_00568] 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 enable conditions are not fulfilled.⌋*(RS_Diag_04192, RS_Diag_04125)*

7.2.2.1.4.5 Dependencies to UDS service 0x85 ControlDTCSettings

[SWS_DM_00088]{DRAFT} **ControlDTCSetting influence (freeze)** ⌈If ControlDTCSetting is set to disabled according to [SWS_DM_00910] 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 or timer remains unchanged.⌋*(RS_Diag_04159, RS_Diag_04125)*

[SWS_DM_00378]{DRAFT} **ControlDTCSetting influence (reset)** ⌈If ControlDTCSetting is set to disabled according to [SWS_DM_00910] 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.⌋*(RS_Diag_04159, RS_Diag_04125)*

7.2.2.2 DTC Status processing

The 'DTC Status processing' is the DMs ability to record and retain UDS status and associated interactions with other SW parts.

7.2.2.2.1 Status processing

'Status processing' is an essential part of the DM functionality. The DM provides means to other SW parts in order to control the UDS DTC status bits.

[SWS_DM_00213]{DRAFT} **DTC status processing** ⌈

The DM shall process the UDS DTC status byte harmonizing with the ISO 14229-1[1] standard.⌋*(RS_Diag_04151)*

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

[SWS_DM_00883]{DRAFT} **UDS DTC status bit transitions triggered by test results** ⌈The DM shall process the UDS DTC status byte triggered by the test results (kPassed or kFailed) reported via the ReportMonitorAction() ([SWS_DM_00543]) function of the corresponding ara::diag::monitor instance ([SWS_DM_00542]). Here, kPassed shall be used as "TestResult [Passed]" and kFailed as "TestResult [Failed]" ([SWS_DM_00541]) as described in [ISO 14229-1] Annex D.2.⌋*(RS_Diag_04151)*

Note that if debouncing for an event is configured, kPrepassed or kPrefailed of enumeration MonitorAction ([SWS_DM_00541]) status reports, reported via ReportMonitorAction, trigger debounce mechanisms (see section 7.2.2.1.4). These status reports do not have direct impact on the UDS DTC status byte. If the status of an event gets fully qualified after debouncing (i.e. kFinallyHealed or kFinallyDefective of enumeration ara::diag::Event::DebouncingState ([SWS_DM_00645])), this information has the same impact on the UDS DTC status byte as if kPassed or kFailed would have been reported via ReportMonitorAction() ([SWS_DM_00543]).

[SWS_DM_00884]{DRAFT} **Resetting the status of the DTC** ⌈If the parameter action in the function ReportMonitorAction ([SWS_DM_00543]) is set to kResetTestFailed ([SWS_DM_00541]), the DM shall update the UDS DTC status byte by setting **only** the kTestFailed bit to FALSE ([SWS_DM_00658]: ara::diag::DTCInformation::UdsStatusBit) and leave all other bits unchanged.⌋*(RS_Diag_04151)*

Rationale: This is an AUTOSAR-specific additional reset condition for the 'testFailed' bit of the UDS DTC status bits.

[SWS_DM_00885]{DRAFT} **UDS DTC status bit transitions triggered by operation cycle changes** ⌈If the function SetOperationCycle() ([SWS_DM_00756]) of the corresponding ara::diag::OperationCycle ([SWS_DM_00751]) instance changes the state of that operation cycle, the DM shall reprocess the UDS DTC status byte according to the operation cycle state change.⌋*(RS_Diag_04178, RS_Diag_04182)*

Note that operation cycles are assigned to events by DiagnosticEventTo-OperationCycleMapping configuration items.

[SWS_DM_00217]{DRAFT} **UDS DTC status bit transitions triggered by ClearDiagnosticInformation UDS service** ⌈If the clearing of a DTC is triggered by the UDS service 0x14 *ClearDiagnosticInformation*, the DM shall process the UDS DTC status byte according to ISO 14229-1[1].⌋*(RS_Diag_04180, RS_Diag_-04067)*

[SWS_DM_00218]{DRAFT} **Trip Counter** ⌈The DM shall take the eventFailure-CycleCounterThreshold configuration parameter as the ConfirmationThreshold value, as defined by ISO 14229-1[1] Annex D.2 . If the TripCounter reaches this ConfirmationThreshold, the DM shall set the UDS DTC status bit kConfirmedDTC to TRUE.⌋*(RS_Diag_04136, RS_Diag_04180, RS_Diag_04157)*

Note that the TripCounter is processed according to the ISO 14229-1[1] Annex D.2 specification.

(In contrast to the TripCounter, the FaultDetectionCounter controls the UDS DTC status bit kTestFailed.)

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

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

7.2.2.2.2 Status change notifications

[SWS_DM_00886]{DRAFT} **Observability of the status byte** ⌈If an AA calls the function GetEventStatus() ([SWS_DM_00649]), the DM shall provide the current status of this event from the corresponding ara::diag::Event instance ([SWS_DM_00646]).⌋*(RS_Diag_04183)*

[SWS_DM_00887]{DRAFT} **Notification about DTC status changes** ⌈If the AA has registered for a DTC status change notification via the function SetEventStatusChangedNotifier() ([SWS_DM_00650]) of the corresponding ara::diag::Event instance ([SWS_DM_00646]), the DM shall call this notifier for each status change of this DTC.⌋*(RS_Diag_04183)*

7.2.2.2.3 Indicators

Indicators can be associated with a particular DTC. Indicators or 'warning outputs' may consist of lamp(s), displayed text information or similar vendor specific expressions.

[SWS_DM_00221]{DRAFT} **Handling indicator status** ⌈The DM shall handle the status of indicators assigned to events by the DiagnosticConnectedIndicator configuration item.⌋*(RS_Diag_04204)*

[SWS_DM_00888]{DRAFT} **Observability of indicator status** ⌈The DM shall provide the status of an indicator via the GetIndicator() function ([SWS_DM_00744]) of the corresponding ara::diag::Indicator instance ([SWS_DM_00741]).⌋*(RS_Diag_04204)*

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

[SWS_DM_00223]{DRAFT} **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.⌋*(RS_Diag_04204)*

For confirmation check [SWS_DM_00218].

[SWS_DM_00224]{DRAFT} **Indicator healing** ⌈The DM shall process indicator healing based on the DiagnosticConnectedIndicator.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.⌋*(RS_Diag_04204)*

7.2.2.2.4 User controlled WarningIndicatorRequest-bit

In some cases (e.g. controlling a failsafe reaction in an application) the WIR-bit (WarningIndicatorRequest-bit) of a corresponding event in DM shall be set/reset by a dedicated "failsafe AA". The "failsafe AA" has to ensure a proper status of the WIR-bit (e.g. regarding to ISO- 14229-1[2] or manufacture specific requirements).

The failsafe AA shall report the required WIR-status to DM (via the function SetLatchedWIRStatus() ([SWS_DM_00652]) of the corresponding ara::diag::event instance ([SWS_DM_00646])) and has to ensure that the current WIR-status of an event (in DM) fits to the current failsafe-status in application:

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

The failsafe AA has to report the status after every change of its failsafe state. Each invocation of the function SetLatchedWIRStatus() ([SWS_DM_00652]) of an ara::diag::event instance ([SWS_DM_00646]) updates the WIR-bit for the corresponding event

Due to not storing the Status-Bit 7 ('warningIndicatorRequested' bit) on Shutdown, the failesafe AA has to ensure that the 'warningIndicatorRequest' bit of an event fits to the current failsafe status after inizalization of the DM.

7.2.2.3 Operation Cycles Management

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

Examples of typical operation cycles are:

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

Operation cycles are managed by the AA, the DM is notified about changes to operation cycle states by using the API interface `ara::diag::OperationCycle::SetOperationCycle` ([SWS_DM_00756]).

[SWS_DM_00889]{DRAFT} **Automatic starting of operation cycles** ⌈If the configuration of `DiagnosticOperationCycle.cycleAutostart` is set to true, the DM shall set the respective state of an `ara::diag::OperationCycle` instance ([SWS_DM_00751]) to `kOperationCycleStart` ([SWS_DM_00750]) during the DM is initializing.⌋*(RS_Diag_04178)*

[SWS_DM_00890]{DRAFT} **Automatic ending of operation cycles** ⌈If the configuration of `DiagnosticOperationCycle.automaticEnd` is set to true, the DM shall set the respective state of an `ara::diag::OperationCycle` instance ([SWS_DM_00751]) to `kOperationCycleEnd` ([SWS_DM_00750]) while the DM is shut down.⌋*(RS_Diag_04178)*

[SWS_DM_00004]{DRAFT} **Operation cycle persistency** ⌈If the configuration of `DiagnosticOperationCycle.cycleStatusStorage` is set to true, the DM shall persist the operation cycle state over the DM shut down.⌋*(RS_Diag_04178)*

[SWS_DM_00891]{DRAFT} **Restart of operation cycles** ⌈If the operation cycle state of an `ara::diag::OperationCycle` instance ([SWS_DM_00751]) was already set to `kOperationCycleStart` ([SWS_DM_00750]) before and the function `SetOperationCycle()` ([SWS_DM_00756]) is called with the value `kOperationCycleStart` ([SWS_DM_00750]), the DM shall restart the operation cycle and perform all steps triggered with a started operation cycle.⌋*(RS_Diag_04178)*

[SWS_DM_00892]{DRAFT} **Operation cycles are only ended once** ⌈If the operation cycle state of an `ara::diag::OperationCycle` instance ([SWS_DM_00751])

was already set to kOperationCycleEnd ([SWS_DM_00750]) before and the function SetOperationCycle() ([SWS_DM_00756]) is called with the value kOperationCycleEnd ([SWS_DM_00750]), the DM shall leave this operation cycle state set to kOperationCycleEnd and take no further actions.⌋*(RS_Diag_04178)*

7.2.2.4 Event memory

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

There can be multiple event memories handled by the DM.

[SWS_DM_00055]{DRAFT} **Supported event memories** ⌈The DM shall support the

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

according to ISO 14229-1[1].⌋*(RS_Diag_04214, RS_Diag_04150)*

[SWS_DM_00911]{DRAFT} **Instances of DTCInformation interface** ⌈The DM shall offer for every configured DiagnosticMemoryDestination a specific instance of the ara::diag::DTCInformation class ([SWS_DM_00657]).⌋*(RS_Diag_04214, RS_Diag_04150)*

[SWS_DM_00056]{DRAFT} **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.⌋*(RS_Diag_04150)*

[SWS_DM_00057]{DRAFT} **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.⌋*(RS_Diag_04214)*

7.2.2.4.1 DTC Introduction

A diagnostic trouble code (DTC) defines a unique identifier mapped to a diagnostic event. The DTC is used by diagnostics, including e.g. UDS communication with an external tester, to uniquely identify data within the event memory database.

[SWS_DM_00060]{DRAFT} **Set of supported DTCs** ⌈The existence of a DiagnosticTroubleCodeUds indicates that the DM shall support this DTC.⌋*(RS_Diag_04201)*

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

7.2.2.4.1.1 Format

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

[SWS_DM_00058]{DRAFT} **DTC interpretation format** ⌈The DM shall use one internal DTC format interpretation that is defined in DiagnosticCommonProps.typeOfDtcSupported.⌋*(RS_Diag_04157)*

Note: Refers to [TPS_DEXT_01008] in [2].

[SWS_DM_CONSTR_00059]{DRAFT} **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

⌋*(RS_Diag_04201)*

7.2.2.4.1.2 Groups

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

A definition of valid DTC groups is provided by ISO 14229-1 [1] - Annex D.1. The DTC group is used in diagnostic just as any other DTC value, the DM internally resolves 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]{DRAFT} **Definition of DTC groups** ⌈The existence of a DiagnosticTroubleCodeGroup shall define the existence of the DTC group with the DTC identifier DiagnosticTroubleCodeGroup.groupNumber⌋*(RS_Diag_04117, RS_Diag_04115)*

Note: Refers to [TPS_DEXT_03014] in [2].

[SWS_DM_00065]{DRAFT} **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 DTC group contains always all configured DTCs.⌋*(RS_Diag_04117)*

[SWS_DM_CONSTR_00082]{DRAFT} **Restriction on the configuration of the DTC group GroupOfAllDTCs** ⌈The DM shall ignore any configuration of a DiagnosticTroubleCodeGroup.groupNumber with a value of 0xFFFFFF.⌋*(RS_Diag_04117)*

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

7.2.2.4.2 Destination

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

[SWS_DM_00083]{DRAFT} **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.⌋*(RS_Diag_04150, RS_Diag_04214)*

[SWS_DM_CONSTR_00084]{DRAFT} **Each DTC shall be assigned to an event memory destination** ⌈The DM shall only support DTCs with a configured DiagnosticTroubleCodeProps.memoryDestination.⌋*(RS_Diag_04150, RS_Diag_04214)*

7.2.2.4.3 EnableConditions

DiagnosticEnableConditions are mapped to DiagnosticEvents by DiagnosticEventToEnableConditionGroupMappings.

[SWS_DM_00568]{DRAFT} **Handling of enable conditions** ⌈If any of the enable conditions mapped to the event are not fulfilled, diag::Monitor::ReportMonitorAction() ([SWS_DM_00543]) shall instantly return without any processing.⌋*(RS_Diag_04192)*

Note: For a regular processing of diag::Monitor::ReportMonitorAction() **all** of the enable conditions mapped to the corresponding event have to be fulfilled.

7.2.2.4.4 DTC related data

The following sections deal with the DTC related data, what includes the triggering and location of freeze frames and extended data records to be stored to. Freeze frames consist of a set of DIDs and extended data records consist of a set of data elements, which shall be stored in configuration dependent situations.

[SWS_DM_00148]{DRAFT} **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)

⌋*(RS_Diag_04211, RS_Diag_04105)*

7.2.2.4.4.1 Triggering for data storage

[SWS_DM_00150]{DRAFT} **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 (see [2]).⌋*(RS_Diag_04211, RS_Diag_04105)*

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

7.2.2.4.4.2 Storage of snapshot record data

[SWS_DM_00151]{DRAFT} **snapshot record numeration** ⌈In case DiagnosticMemoryDestination.typeOfFreezeFrameRecordNumeration is set to calculated, the DM shall store freeze frames numbered consecutively starting with 1 in their chronological order. If the parameter is set to configured, the DM shall store the records based on the DiagnosticFreezeFrame.recordNumber configuration parameters of the respective freeze frames.⌋*(RS_Diag_04205, RS_Diag_04189)*

[SWS_DM_00152]{DRAFT} **Number of snapshot records for a DTC** ⌈In case DiagnosticMemoryDestination.typeOfFreezeFrameRecordNumeration is set to calculated, the number of snapshot record the DM is able to store for a DTC shall be determined by the DiagnosticTroubleCodeProps.maxNumberFreezeFrameRecords configuration parameter. In case DiagnosticMemoryDestination.typeOfFreezeFrameRecordNumeration is set to configured, the number of snapshot recordss is determined by the number of DiagnosticFreezeFrames configured for a DTC.⌋*(RS_Diag_04205, RS_Diag_04190)*

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

[SWS_DM_00893]{DRAFT} **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.snapshotRecordContent configuration class. Each referenced DiagnosticDataIdentifier shall be captured in its order via the diag::GenericDataIdentifier::Read() function ([SWS_DM_00636]) or diag::DataIdentifier::Read() ([SWS_DM_00640]) according to its PortPrototype mapping.⌋*(RS_Diag_04205, RS_Diag_04127)*

[SWS_DM_00894]{DRAFT} **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 and there is a registered update notification via the function SetSnapshotRecordUpdatedNotifier() of the corresponding ara::diag::DTCInformation instance ([SWS_DM_00668]), the DM shall call this notifier for each snapshot record update.⌋*(RS_Diag_04148)*

7.2.2.4.4.3 Storage of extended data

[SWS_DM_00154]{DRAFT} **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.⌋*(RS_Diag_04206, RS_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]{DRAFT} **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.⌋*(RS_Diag_04206, RS_Diag_04189)*

[SWS_DM_00895]{DRAFT} **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 Read() function of the ara::diag::DataElement instance ([SWS_DM_00596]).⌋(*RS_Diag_04206, RS_Diag_04127*)

7.2.2.4.5 Clearing DTCs

Clearing a DTC or a DTC group is the ability of the DM to reset the UDS DTC status byte of each DTC and deleting DTC assigned snapshot records, extended data records and further DTC-related data.

[SWS_DM_00116]{DRAFT} **Clearing a DTC group** ⌈When the DM is about to clear a DTC group it shall apply the same clear operation process as for a single DTC on all the DTCs of the DTC group which is cleared.⌋(*RS_Diag_04117*)

[SWS_DM_00117]{DRAFT} **Clearing a DTC** ⌈When the DM is about to clear a DTC it shall reset the event and UDS DTC status byte and clear the snapshot records and extended data records stored for this DTC and its DTC-related data.⌋(*RS_Diag_04117*)

7.2.2.4.5.1 Locking of the DTC clearing process by a client

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

[SWS_DM_00144]{DRAFT} **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.⌋(*RS_Diag_04117*)

[SWS_DM_00145]{DRAFT} **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.⌋(*RS_Diag_04117*)

[SWS_DM_00146]{DRAFT} **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.⌋(*RS_Diag_04117*)

[SWS_DM_00147]{DRAFT} **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).⌋(*RS_Diag_04117*)

7.2.2.4.5.2 ClearConditions

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

[SWS_DM_00896]{DRAFT} **Handling of DiagnosticClearConditions** ⌈If any of the clear conditions mapped to the DTC to be cleared are not fulfilled by a call of the function SetCondition() from an ara::diag::Condition instance ([SWS_DM_00715]) with the value kConditionFalse ([SWS_DM_00710]), the clear is forbidden. Otherwise (all of the clear conditions mapped to the DTC are fulfilled) the clear is allowed.⌋*(RS_Diag_04117)*

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

[SWS_DM_00123]{DRAFT} **Block clearing of UDS DTC status byte during a clear DTC operation** ⌈If the DM is requested to clear a DTC with a forbidden clear according to [SWS_DM_00896] and a DiagnosticEventToTroubleCodeUdsMapping exists with a mapping from this DTC to an event and the event has DiagnosticEvent.clearEventBehavior set to noStatusByteChange, the DM shall not change the UDS DTC status byte.⌋*(RS_Diag_04117)*

[SWS_DM_00124]{DRAFT} **Limited clearing of UDS DTC status byte during a clear DTC operation** ⌈If the DM is requested to clear a DTC with a forbidden clear according to [SWS_DM_00896] and a DiagnosticEventToTroubleCodeUdsMapping exists with a mapping from this DTC to an event and the event has DiagnosticEvent.clearEventBehavior set to onlyThisCycleAndReadiness, the DM shall set the following UDS DTC status bits:

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

and leave all other bits unchanged.⌋*(RS_Diag_04117)*

[SWS_DM_00121]{DRAFT} **Forbidden clearing of snapshot records and extended data records** ⌈If the DM is requested to clear a DTC with a forbidden clear according to [SWS_DM_00896] the DM shall leave all snapshot records and extended data records for this DTC unchanged.⌋*(RS_Diag_04117)*

7.2.2.4.5.3 DTC clearing triggered by application

Besides the UDS request ClearDiagnosticInformation according to section 7.2.1.6.5.1 the DM supports the use case that the fault memory is cleared by an application call. One of the use cases is clearing of user-defined event memory for diagnostic implementation without the ISO 14229-1[1] extension as described in section 7.2.1.6.5.1.

This could be realized using a dedicated diagnostic routine service, whose application is in charge of the clearing process.

[SWS_DM_00262]{DRAFT} **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.⌋*(RS_Diag_04194)*

[SWS_DM_00897]{DRAFT} **Usage of ClearDTC Interface** ⌈If the function Clear() of the ara::diag::DTCInformation instance ([SWS_DM_00671]) is called, the DM shall clear the DTC or DTC group provided in the parameter DTC-Group (compare function declaration ara::diag::DTCInformation::Clear(); ([SWS_DM_00671])). The clear DTC shall clear the fault memory associated to the instance of the ara::diag::DTCInformation class only.⌋*(RS_Diag_04194)*

[SWS_DM_00898]{DRAFT} **ClearDTC call on invalid DTC or DTC group** ⌈If the function Clear() of the ara::diag::DTCInformation instance ([SWS_DM_00671]) is called and the parameter DTCGroup of the function Clear() has no matching configured DTC group according to [SWS_DM_00064] or configured DTC by DiagnosticTroubleCodeUds.udsDtcValue, the DM shall trigger the error kWrongDtc for that function call and the DM shall return without any further action.⌋*(RS_Diag_04194)*

[SWS_DM_00899]{DRAFT} **ClearDTC called while another clear operation is in progress** ⌈If the function Clear() of the ara::diag::DTCInformation instance ([SWS_DM_00671]) is called and another clear DTC operation is currently in progress, the DM shall trigger the error kBusy.⌋*(RS_Diag_04194)*

[SWS_DM_00900]{DRAFT} **ClearDTC processing in case of memory errors** ⌈If the function Clear() of the ara::diag::DTCInformation instance ([SWS_DM_00671]) is called and the DM receives physical memory errors upon its access to the Non-volatile Memory and thus cannot guarantee that the clear operation was done successfully, the DM shall trigger the error kMemoryError.⌋*(RS_Diag_04194)*

[SWS_DM_00901]{DRAFT} **Possible failure of ClearDTC** ⌈If the function Clear() of the ara::diag::DTCInformation instance ([SWS_DM_00671]) is called and the clear operation fails due to the reasons according to [SWS_DM_00122], the DM shall trigger the error kFailed.⌋*(RS_Diag_04194)*

7.2.2.4.6 Aging

A stored DTC can age in terms of reaching a threshold value of passed operation cycles, specified by the vendor, where no failed tests have been reported by a monitoring application. The amount of operation cycles, where these non-failed reports occur is called the Aging counter. After the threshold is reached, the DTC is cleared from the event memory.

[SWS_DM_00237]{DRAFT} **Aging** ⌈The DM shall only support Aging for DTCs, if the corresponding DiagnosticTroubleCodeProps.agingAllowed configuration parameter is set.⌋*(RS_Diag_04133)*

[SWS_DM_00238]{DRAFT} **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 (according to [SWS_DM_00224]) is completed ('warningIndicatorRequested' bit is set to 0).⌋*(RS_Diag_04133)*

[SWS_DM_00239]{DRAFT} **Aging counter** ⌈The DM shall support an Aging counter for each event memory entry.⌋*(RS_Diag_04133)*

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

[SWS_DM_00240]{DRAFT} **Processing the Aging counter** ⌈The DM shall only allow processing the Aging counter if the related DTC is stored in the event memory, the status is qualified as passed ('testFailed' bit is set to 0) and healing, according to [SWS_DM_00238], is fulfilled.⌋*(RS_Diag_04133)*

[SWS_DM_00241]{DRAFT} **Aging cycle and threshold** ⌈The Aging shall be calculated based on the referred DiagnosticOperationCycle via the reference DiagnosticAging.agingCycle. The DiagnosticAging.threshold defines the number of Aging cycles until Aging. If DiagnosticCommonProps.agingRequiresTestedCycle is set, the cycle shall only be considered in which the status was reported but not failed ('testNotCompletedThisOperationCycle' bit and 'testFailedThisOperationCycle' bit are set to 0). If the threshold is reached, the event memory entry shall be deleted (aged) from the event memory.⌋*(RS_Diag_04133)*

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

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

⌋*(RS_Diag_04140)*

[SWS_DM_00242]{DRAFT} **Re-occurrence after Aging** ⌈The DM shall treat the re-occurrence of previously aged events like new events, since they were previously deleted from the event memory by Aging. This corresponds to all DTC-related data (i.e. counters, thresholds, etc.) being reset to their initial values.⌋*(RS_Diag_04133)*

7.2.2.4.7 NumberOfStoredEntries

[SWS_DM_00902]{DRAFT} **NumberOfStoredEntries** ⌈If the function GetNumberOfStoredEntries() from the ara::diag::DTCInformation instance

([SWS_DM_00669]) is called, the DM shall return the number of event memory entries (DTCs) currently stored in this event memory, where the status of a DTC is pendingDTC = 1 and/or confirmedDTC = 1. An update notification shall be sent to the function registered via SetNumberOfStoredEntriesNotifier() ([SWS_DM_00670]) whenever the value of NumberOfStoredEntries has changed.⌋*(RS_Diag_04109)*

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

7.2.3 Required Configuration

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

[SWS_DM_CONSTR_00168]{DRAFT} **Required operation cycles for diagnostic events** ⌈Each DiagnosticEvent requires exactly one DiagnosticEventToOperationCycleMapping referencing the diagnosticEvent and one DiagnosticOperationCycle.⌋*(RS_Diag_04178)*

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

7.2.4 Diagnostic Data Management

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

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

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

7.2.4.1 Internal and External Diagnostic Data Elements

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

Table 7.4 gives a list of the supported internal DiagnosticDataElements, where

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

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 Conversation	1 byte	DCM
DCM_SECURITY_LEVEL	Current security level of contextual Diagnostic Conversation	1 byte	DCM

Table 7.4: Supported internal DiagnosticDataElements

[SWS_DM_00393]{DRAFT} **Retrieving data for internal DiagnosticDataElements** ⌈If DM requires to provide or store data configured as internal DiagnosticDataElement which is supported by the Diagnostic Server instance according to Table 7.4, then DM shall use the respective internally managed data value as defined in Table 7.4.⌋*(RS_Diag_04097)*

[SWS_DM_CONSTR_00394]{DRAFT} **Internal DiagnosticDataElements are read-only** ⌈A DiagnosticDataIdentifier referenced by a DiagnosticWriteDataByIdentifier service shall not contain any internal DiagnosticDataElement.⌋*(RS_Diag_04097)*

An internal DiagnosticDataElement is called DCM-exclusive resp. DEM-exclusive if the context of the name token described in Table 7.4 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 7.4 might in future be extended by internal DiagnosticDataElements not restricted to exclusive use within a DCM resp. DEM context.

[SWS_DM_CONSTR_00395]{DRAFT} **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.⌋*(RS_Diag_04097)*

[SWS_DM_CONSTR_00396]{DRAFT} **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.⌋*(RS_Diag_04097)*

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

[SWS_DM_00905]{DRAFT} **Retrieving data for external DiagnosticDataElements** ⌈If the Diagnostic Server instance is required to read data configured as external DiagnosticDataElement, then the Diagnostic Server instance shall utilize the associated RPortPrototype typed by the DataElement

class ([SWS_DM_00603]) and call its DataElement::Read ([SWS_DM_00596]) function.⌋ *(RS_Diag_04097)*

Note: In general, there are multiple instances of DataElement class ([SWS_DM_00603]) available in the running system. Which instance to choose for the given request to read an external DiagnosticDataElement is part of system integration. Support for this integration is provided by DiagnosticMappings described in section 7.2.4.2.1.

7.2.4.2 Reading and Writing Diagnostic Data Identifier

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

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

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

7.2.4.2.1 Supported Diagnostic Mappings

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

DiagnosticMapping	**diagnostics endpoint**	**target endpoint**
DiagnosticProvidedDataMapping	DiagnosticDataElement	DM internal data provider
DiagnosticServiceDataMapping	DiagnosticDataElement	DataPrototype
DiagnosticServiceSwMapping	DiagnosticDataElement	SwcServiceDependency
DiagnosticServiceDataIdentifierPortMapping	DiagnosticDataIdentifier	SwcServiceDependency

Table 7.5: Diagnostic Mappings

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

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

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

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

The DiagnosticServiceDataIdentifierPortMapping maps a DiagnosticDataByIdentifier in the role of diagnosticDataIdentifier to a SwcServiceDependency in the role of swcServiceDependencyInExecutable.

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

7.2.4.2.2 Reading Diagnostic Data Identifier

[SWS_DM_00401]{DRAFT} **Reading Diagnostic Data Identifier on Data Element level** ⌈If the Diagnostic Server instance is required to read data configured as DiagnosticDataIdentifier and at least on of the DiagnosticDataElements aggregated in this DiagnosticDataIdentifier is referenced by some DiagnosticMapping, then Diagnostic Server instance shall retrieve the data by reading data from each DiagnosticDataElement separately according to [SWS_DM_00393] and [SWS_DM_00905].⌋*(RS_Diag_04097)*

[SWS_DM_00848]{DRAFT} **Reading Diagnostic Data Identifier by DataIdentifier interface** ⌈If the Diagnostic Server instance is required to read data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticServiceDataIdentifierPortMapping of category DATA_IDENTIFIER, then the Diagnostic Server instance shall use the DataIdentifier class ([SWS_DM_00601]) or diag::GenericDataIdentifier class ([SWS_DM_00607] instance according to its PortPrototype mapping and associated to the DiagnosticDataIdentifier for reading the data.⌋*(RS_Diag_04097)*

[SWS_DM_00849]{DRAFT} **Reading Diagnostic Data Identifier by GenericUDSService interface** ⌈If the Diagnostic Server instance is required to read data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticServiceDataIdentifierPortMapping of category GENERIC_

UDS_SERVICE, then the Diagnostic Server instance shall use the instance of the diag::GenericUDSService class ([SWS_DM_00602]) referenced by the DiagnosticServiceDataIdentifierPortMapping and call its diag::GenericUDSService::HandleMessage ([SWS_DM_00618]) method with sid parameter set to 0x22 and request_data set to the id of the DiagnosticDataIdentifier. The diag::GenericUDSService::OperationOutput ([SWS_DM_00578]) is respectively composed of the requested id and the content of every diag::DataElement::OperationOutput ([SWS_DM_00580]) of the related dataElement.⌋*(RS_Diag_04097)*

[SWS_DM_00850]{DRAFT} **Default Service Interface for reading DiagnosticDataIdentifier** ⌈If the Diagnostic Server instance is required to read data configured as DiagnosticDataIdentifier and none of the requirements [SWS_DM_00401], [SWS_DM_00848], [SWS_DM_00849] applies, then the Diagnostic Server instance shall utilize the associated RPortPrototype typed by the DataIdentifier class ([SWS_DM_00601]) and call its DataIdentifier::Read ([SWS_DM_00640]) function.⌋*(RS_Diag_04097)*

Note: The default configuration as described in [SWS_DM_00850] assumes, that there is a single instance of PPortPrototype defined in the system, matching the RPortPrototype associated to the requested DiagnosticDataIdentifier. In this case, it is part of integration step to link these two ports.

7.2.4.2.3 Writing Diagnostic Data Identifier

[SWS_DM_00906]{DRAFT} **Writing Diagnostic Data Identifier by DataIdentifier interface** ⌈If the Diagnostic Server instance is required to write data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticServiceDataIdentifierPortMapping of category DATA_IDENTIFIER, then the Diagnostic Server instance shall use the diag::GenericDataIdentifier class ([SWS_DM_00607]) or DataIdentifier class ([SWS_DM_00601]) according to its PortPrototype mapping and associated to the DiagnosticDataIdentifier for writing the data.⌋*(RS_Diag_04097)*

[SWS_DM_00908]{DRAFT} **Writing Diagnostic Data Identifier by GenericUDSService interface** ⌈If the Diagnostic Server instance is required to writing data configured as DiagnosticDataIdentifier which is referenced by a DiagnosticServiceDataIdentifierPortMapping of category GENERIC_UDS_SERVICE, then the Diagnostic Server instance shall use the instance of the diag::GenericUDSService class referenced by the DiagnosticServiceDataIdentifierPortMapping and call its diag::GenericUDSService::HandleMessage ([SWS_DM_00618]) with SID set to 0x2E and request_data set to the id of this DiagnosticDataIdentifier followed by the data to be written to this DiagnosticDataIdentifier.⌋ *(RS_Diag_04097)*

[SWS_DM_00907]{DRAFT} **Default Service Interface for writing `DiagnosticDataIdentifier`** ⌈If the `Diagnostic Server instance` is required to write data configured as `DiagnosticDataIdentifier` and none of the requirements [SWS_DM_00906], [SWS_DM_00908] applies, then the `Diagnostic Server instance` shall utilize the associated `RPortPrototype` typed by the `DataIdentifier class` ([SWS_DM_00601]) and call `DataIdentifier::Write()` ([SWS_DM_00598]).⌋*(RS_Diag_04097)*

Note: The default configuration as described in [SWS_DM_00907] assumes, that there is a single instance of `PPortPrototype` defined in the system matching the `RPortPrototype` associated to the requested `DiagnosticDataIdentifier`. In this case, it is part of integration step to link these two ports.

7.2.4.2.4 Reading and writing VIN data

[SWS_DM_00903]{DRAFT} **Reading DiagnosticDataIdentifier configured for representing `VIN`** ⌈If the `Diagnostic Server instance` needs to read data configured as `DiagnosticDataIdentifier` with attribute `representsVin` set to `true`, the `Diagnostic Server instance` shall obtain it by using *diag::GenericDataIdentifier::Read()* ([SWS_DM_00636]) or *diag::DataIdentifier::Read()* ([SWS_DM_00640]) according to its PortPrototype mapping.⌋*(RS_Diag_04097)*

[SWS_DM_00904]{DRAFT} **Writing DiagnosticDataIdentifier configured for representing `VIN`** ⌈If the `Diagnostic Server instance` needs to write data configured as `DiagnosticDataIdentifier` with attribute `representsVin` set to `true`, the `Diagnostic Server instance` shall call *diag::GenericDataIdentifier::Write()* ([SWS_DM_00637]) or *diag::DataIdentifier::Write()* ([SWS_DM_00598]) according to its PortPrototype mapping.⌋*(RS_Diag_04097)*

8 API specification

This chapter lists all provided and required C++ API interfaces of the DM. The C++ API interfaces are divided into two parts:

- UDS Transportlayer interface
  A plug-in interface to extend the DM by own transport layers
- Diagnostic Application interface
  A DiagnosticPortInterfaces is representing a corresponding code instance. The deployment is simplified due to a direct mapping to DiagnosticObject in DEXT.

8.1 C++ UDS Transportlayer API Interfaces

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

8.1.1 UDS Transportlayer Types

8.1.1.1 uds_transport::ByteVector

[SWS_DM_00338]{DRAFT} ⌈

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

⌋()

8.1.1.2 uds_transport::ChannelID

[SWS_DM_00337]{DRAFT} ⌈

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

⌋()

8.1.1.3 uds_transport::Priority

[SWS_DM_00451]{DRAFT} ⌈

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

⌋()

8.1.1.4 uds_transport::ProtocolKind

[SWS_DM_00452]{DRAFT} ⌈

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

⌋()

8.1.1.5 uds_transport::UdsMessageConstPtr

[SWS_DM_00304]{DRAFT} ⌈

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

⌋()

8.1.1.6 uds_transport::UdsMessagePtr

[SWS_DM_00303]{DRAFT} ⌈

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

⌋()

8.1.1.7 uds_transport::UdsTransportProtocolHandlerID

[SWS_DM_00336]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandlerID
Scope:	namespace ara::diag::uds_transport
Derived from:	typedef uint8_t

▽

△

Syntax:	`using ara::diag::uds_transport::UdsTransportProtocolHandlerID = uint8_t;`
Header file:	#include "ara/diag/uds_transport/protocol_types.h"
Description:	UdsTransportProtocolHandler are flexible "plugins", which need an identification.

⌋()

8.1.2 UdsMessage Class

[SWS_DM_00291]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::uds_transport::UdsMessage
Scope:	namespace ara::diag::uds_transport
Syntax:	`class UdsMessage {...};`
Header file:	#include "ara/diag/uds_transport/uds_message.h"
Description:	class represents an UDS message exchanged between DM generic core (UdsTransport ProtocolMgr) and a specific implementation of UdsTransportProtocolHandler on diagnostic request reception path or diagnostic response transmission path. UdsMessage provides the storage for UDS requests/responses. Instances of UdsMessage (with optimized resource allocation) are only created by DM generic core. UdsTransport ProtocolHandler read/write on it.

⌋()

8.1.2.1 Types

8.1.2.1.1 uds_transport::UsdMessage::Address

[SWS_DM_00293]{DRAFT} ⌈

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

⌋()

8.1.2.1.2 uds_transport::UsdMessage::MetaInfoMap

[SWS_DM_00294]{DRAFT} ⌈

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

⌋(*RS_Diag_04170*)

8.1.2.1.3 uds_transport::UsdMessage::TargetAddressType

[SWS_DM_00296]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::uds_transport::UdsMessage::TargetAddressType	
Scope:	class ara::diag::uds_transport::UdsMessage	
Underlying type:	std::uint8_t	
Syntax:	`enum class TargetAddressType : std::uint8_t {...};`	
Values:	kPhysical= 0	–
	kFunctional= 1	–
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	type of target address in UdsMessage	

⌋()

8.1.2.2 Methods

8.1.2.2.1 uds_transport::UdsMessage::UdsMessage

[SWS_DM_09012]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::uds_transport::UdsMessage::UdsMessage()
Scope:	class ara::diag::uds_transport::UdsMessage
Visibility:	protected

▽

△

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

⌋()

8.1.2.2.2 uds_transport::UdsMessage::UdsMessage

[SWS_DM_09011]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::uds_transport::UdsMessage::UdsMessage(const UdsMessage &other)	
Scope:	class ara::diag::uds_transport::UdsMessage	
Visibility:	protected	
Syntax:	`UdsMessage (const UdsMessage &other)=default;`	
Parameters (in):	other	Object to copy-construct from
Thread Safety:	reentrant	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Copy constructing the uds message.	

⌋()

8.1.2.2.3 uds_transport::UdsMessage::UdsMessage

[SWS_DM_09013]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::uds_transport::UdsMessage::UdsMessage(UdsMessage &&other)	
Scope:	class ara::diag::uds_transport::UdsMessage	
Visibility:	protected	
Syntax:	`UdsMessage (UdsMessage &&other) noexcept=default;`	
Parameters (in):	other	Object to move-construct from
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Move constructing the uds message.	

⌋()

8.1.2.2.4 uds_transport::UdsMessage::UdsMessage::operator=

[SWS_DM_09014]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::uds_transport::UdsMessage::operator=(const UdsMessage &other)	
Scope:	class ara::diag::uds_transport::UdsMessage	
Visibility:	protected	
Syntax:	`UdsMessage& operator= (const UdsMessage &other)=default;`	
Parameters (in):	other	Object to copy-assign from.
Return value:	UdsMessage &	–
Thread Safety:	reentrant	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Copy assigning the uds message.	

⌋()

8.1.2.2.5 uds_transport::UdsMessage::UdsMessage::operator=

[SWS_DM_09018]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::uds_transport::UdsMessage::operator=(UdsMessage &&other)	
Scope:	class ara::diag::uds_transport::UdsMessage	
Visibility:	protected	
Syntax:	`UdsMessage& operator= (UdsMessage &&other) noexcept=default;`	
Parameters (in):	other	Object to move-assign from.
Return value:	UdsMessage &	–
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Move assigning the uds message.	

⌋()

8.1.2.2.6 uds_transport::UdsMessage::~UdsMessage

[SWS_DM_09010]{DRAFT} ⌈

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

⌋()

8.1.2.2.7 uds_transport::UdsMessage::AddMetaInfo

[SWS_DM_00302]{DRAFT} ⌈

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

⌋(*RS_Diag_04170*)

8.1.2.2.8 uds_transport::UdsMessage::GetPayload

[SWS_DM_00300]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::uds_transport::UdsMessage::GetPayload()	
Scope:	class ara::diag::uds_transport::UdsMessage	
Syntax:	`virtual const uds_transport::ByteVector& GetPayload () const;`	
Return value:	const uds_transport::ByteVector &	The entire payload (A_Data)
Thread Safety:	unsafe	
Header file:	#include "ara/diag/uds_transport/uds_message.h"	
Description:	Get the UDS message data starting with the SID (A_Data as per ISO)	

▽

△

Notes:	marked as "unsafe" with regard to threadsafety as implementation is allowed to do ressource allocation of buffer in the context of this call.

⌋()

[SWS_DM_00301]{DRAFT} ⌈

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

⌋()

8.1.2.2.9 uds_transport::UdsMessage::GetSa

[SWS_DM_00297]{DRAFT} ⌈

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

⌋()

8.1.2.2.10 uds_transport::UdsMessage::GetTa

[SWS_DM_00298]{DRAFT} ⌈

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

⌋()

8.1.2.2.11 uds_transport::UdsMessage::GetTaType

[SWS_DM_00299]{DRAFT} ⌈

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

⌋()

8.1.3 UdsTransportProtocolHandler Class

[SWS_DM_00315]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler
Scope:	namespace ara::diag::uds_transport
Syntax:	`class UdsTransportProtocolHandler {...};`
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"
Description:	Abstract Class, which a specific UDS Transport Protocol (plugin) shall subclass.

⌋()

8.1.3.1 Types

8.1.3.1.1 uds_transport::UdsTransportProtocolHandler::InitializationResult

[SWS_DM_09017]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::InitializationResult	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler	
Underlying type:	std::uint8_t	
Syntax:	`enum class InitializationResult : std::uint8_t {...};`	
Values:	kInitializeOk= 0	–
	kInitializeFailed= 1	–
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"	
Description:	Result of Initialize handler.	

⌋()

8.1.3.2 Methods

8.1.3.2.1 uds_transport::UdsTransportProtocolHandler::UdsTransportProtocolHandler

[SWS_DM_09015]{DRAFT} ⌈

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

⌋()

8.1.3.2.2 uds_transport::UdsTransportProtocolHandler::~UdsTransport

[SWS_DM_09016]{DRAFT} ⌈

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

⌋()

8.1.3.2.3 uds_transport::UdsTransportProtocolHandler::GetHandlerID

[SWS_DM_00325]{DRAFT} ⌈

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

⌋()

8.1.3.2.4 uds_transport::UdsTransportProtocolHandler::Initialize

[SWS_DM_00319]{DRAFT} ⌈

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

⌋()

8.1.3.2.5 uds_transport::UdsTransportProtocolHandler::NotifyReestablishment

[SWS_DM_00326]{DRAFT} ⌈

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

⌋()

8.1.3.2.6 uds_transport::UdsTransportProtocolHandler::Start

[SWS_DM_00322]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::uds_transport::UdsTransportProtocolHandler::Start()
Scope:	class ara::diag::uds_transport::UdsTransportProtocolHandler
Syntax:	`virtual void Start ()=0;`
Return value:	None
Header file:	#include "ara/diag/uds_transport/protocol_handler.h"
Description:	Start processing the implemented Uds Transport Protocol. The implementation shall call its superclass Start() method as there might be some stack specific implementation. Implementation shall be asynchronous as DM might start many/ different UdsTransportProtocolHandler in parallel and strong serialization of all those starts just unnecessarily slows down DM startup.

⌋()

8.1.3.2.7 uds_transport::UdsTransportProtocolHandler::Stop

[SWS_DM_00323]{DRAFT} ⌈

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

⌋()

8.1.3.2.8 uds_transport::UdsTransportProtocolHandler::Transmit

[SWS_DM_00327]{DRAFT} ⌈

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

▽

△

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

⌋()

8.1.4 UdsTransportProtocolMgr Class

[SWS_DM_00306]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr
Scope:	namespace ara::diag::uds_transport
Syntax:	`class UdsTransportProtocolMgr {...};`
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"
Description:	

⌋()

8.1.4.1 Types

8.1.4.1.1 uds_transport::UdsTransportProtocolMgr::GlobalChannelIdentifier

[SWS_DM_09021]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::GlobalChannelIdentifier
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr
Derived from:	std::tuple<UdsTransportProtocolHandlerID, ChannelID>
Syntax:	`using ara::diag::uds_transport::UdsTransportProtocolMgr::GlobalChannel Identifier = std::tuple<UdsTransportProtocolHandlerID, ChannelID>;`
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"

▽

△

Description:	Type of tuple to pack UdsTransportProtocolHandlerID and ChannelID together, to form a global unique (among all used UdsTransportProtocolHandlers within DM) identifier of a UdsTransport Protocol channel.

⌋()

8.1.4.1.2 uds_transport::UdsTransportProtocolMgr::IndicationResult

[SWS_DM_00384]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::IndicationResult	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Underlying type:	std::uint8_t	
Syntax:	`enum class IndicationResult : std::uint8_t {...};`	
Values:	kIndicationOk= 0	–
	kIndicationOccupied= 1	–
	kIndicationOverflow= 2	–
	kIndicationUnknownTargetAddress= 3	–
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:		

⌋()

8.1.4.1.3 uds_transport::UdsTransportProtocolMgr::TransmissionResult

[SWS_DM_00307]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::TransmissionResult	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Underlying type:	std::uint8_t	
Syntax:	`enum class TransmissionResult : std::uint8_t {...};`	
Values:	kTransmitOk= 0	–
	kTransmitFailed= 1	–
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:		

⌋()

8.1.4.2 Methods

8.1.4.2.1 uds_transport::UdsTransportProtocolMgr::ChannelReestablished

[SWS_DM_00313]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::ChannelReestablished(GlobalChannelIdentifier global_channel_id)	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Syntax:	`virtual void ChannelReestablished (GlobalChannelIdentifier global_channel_id)=0;`	
Parameters (in):	global_channel_id	transport protocol channel, which is available again.
Return value:	None	
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:	notification call from the given transport channel, that it has been reestablished since the last (Re)Start from the UdsTransportProtocolHandler to which this channel belongs. To activate this notification a previous call to UdsTransportProtocolHandler::NotifyReestablishment() has to be done. See further documentation at UdsTransportProtocolHandler::NotifyReestablishment().	

⌋()

8.1.4.2.2 uds_transport::UdsTransportProtocolMgr::HandleMessage

[SWS_DM_00311]{DRAFT} ⌈

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

⌋()

8.1.4.2.3 uds_transport::UdsTransportProtocolMgr::HandlerStopped

[SWS_DM_00314]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::uds_transport::UdsTransportProtocolMgr::HandlerStopped(UdsTransportProtocol HandlerID handler_id)	
Scope:	class ara::diag::uds_transport::UdsTransportProtocolMgr	
Syntax:	`virtual void HandlerStopped (UdsTransportProtocolHandlerID handler_ id)=0;`	
Parameters (in):	handler_id	indication, which plugin stopped.
Return value:	None	
Header file:	#include "ara/diag/uds_transport/protocol_mgr.h"	
Description:	notification from handler, that it has stopped now (e.g. closed down network connections, freed resources, etc...) This callback is expected as a reaction from handler to a call to UdsTransportProtocol Handler::Stop.	

⌋()

8.1.4.2.4 uds_transport::UdsTransportProtocolMgr::IndicateMessage

[SWS_DM_00309]{DRAFT} ⌈

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

⌋()

8.1.4.2.5 uds_transport::UdsTransportProtocolMgr::NotifyMessageFailure

[SWS_DM_00310]{DRAFT} ⌈

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

⌋()

8.1.4.2.6 uds_transport::UdsTransportProtocolMgr::TransmitConfirmation

[SWS_DM_00312]{DRAFT} ⌈

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

⌋()

8.1.5 Sequence Diagramms of UDS Transport Layer Interaction

8.1.5.1 Lifecycle

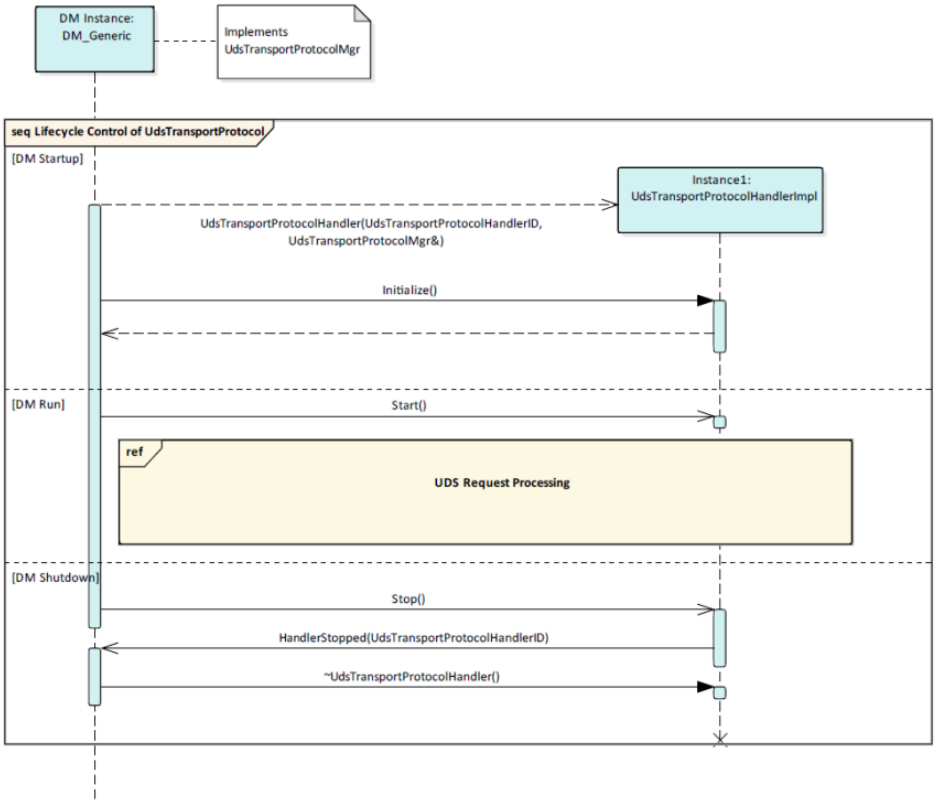

Figure 8.1: UDS Transport Lifecycle

8.1.5.2 UDS Request Processing

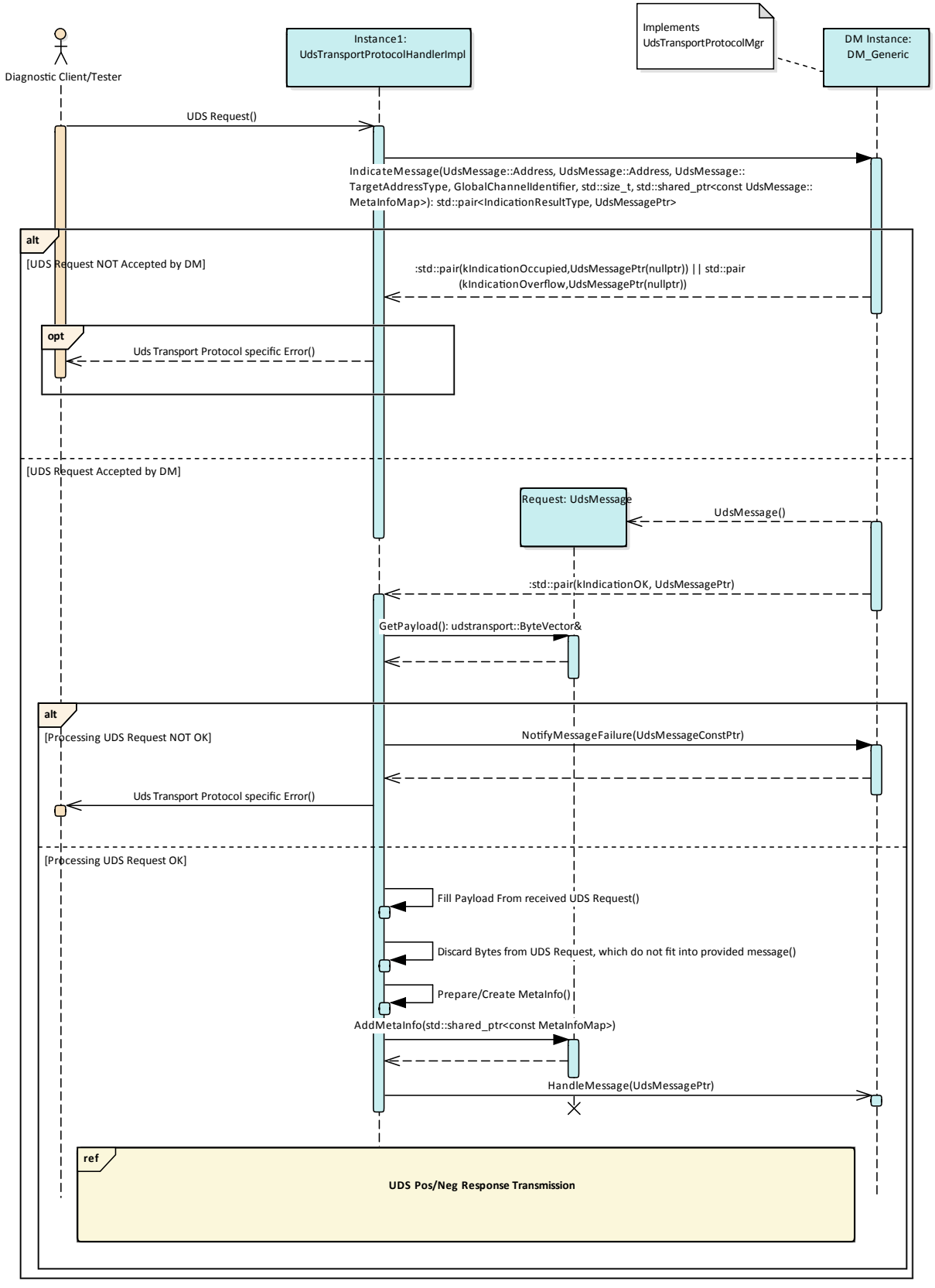

Figure 8.2: UDS Transport Request Processing

8.1.5.3 UDS Response Transmission

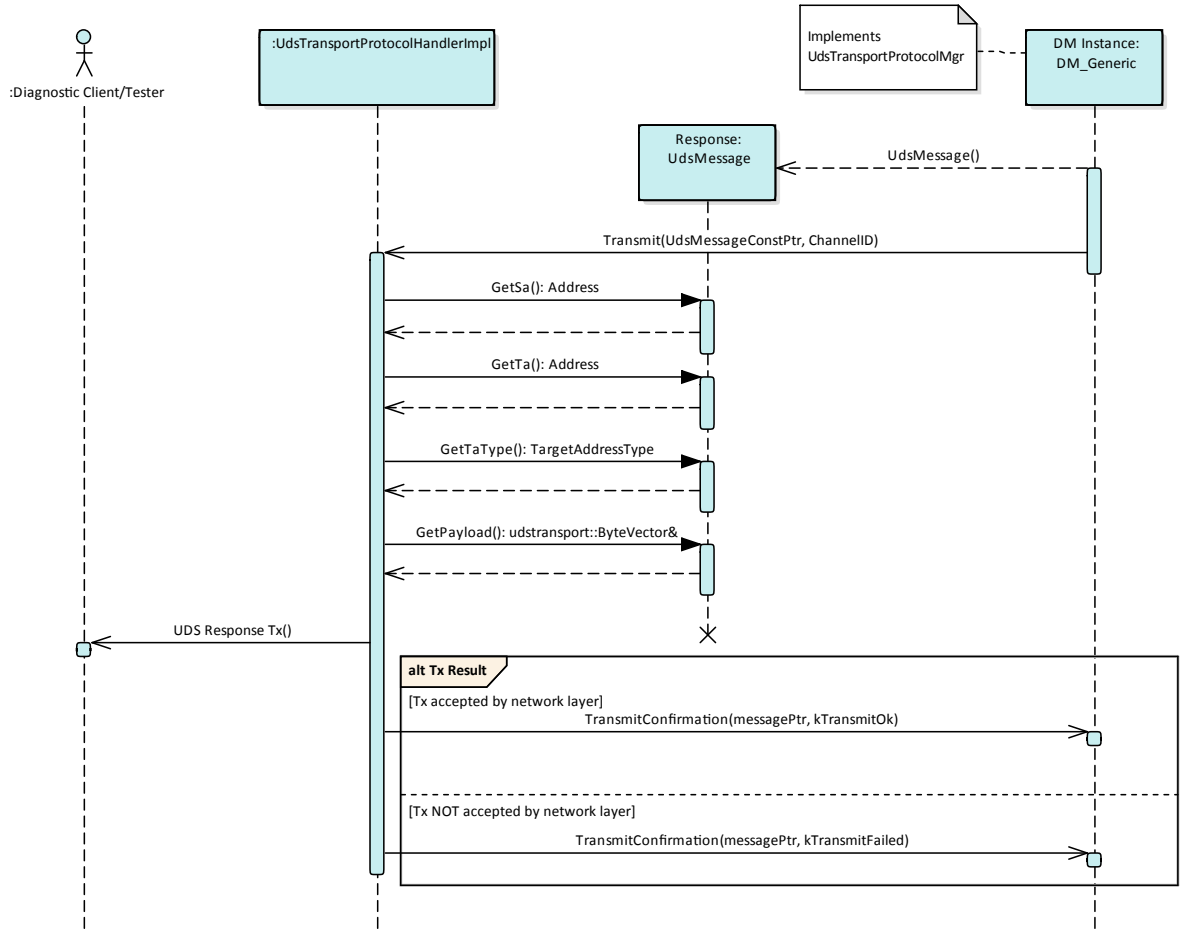

Figure 8.3: UDS Response Transmission

8.1.5.4 Channel Reestablishment

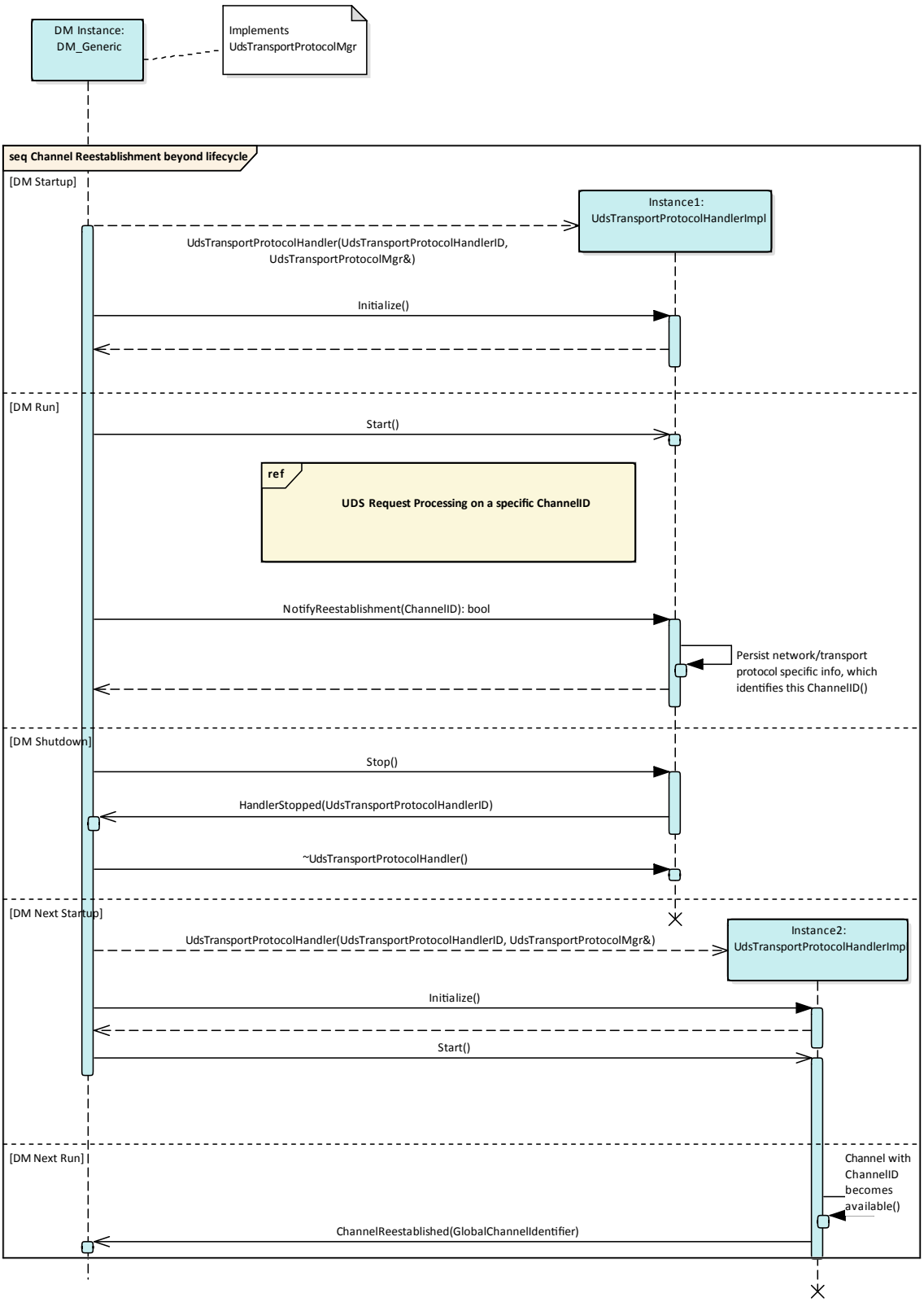

Figure 8.4: UDS Transport Channel Reestablishment

8.2 C++ Diagnostic API Interfaces

This chapter lists all provided and required C++ API interfaces of the DM for interaction with application.

8.2.1 Introduction

Specialized PortInterfaces (DiagnosticPortInterfaces) allow an optimized deployment in the integration. In comparison of a regular ServiceInterface where each interface instance could be deployed individually, the diagnostic PortInterfaces can only be deployed for a complete process.

[SWS_DM_00561]{DRAFT} **Deployment of diagnostic PortInterfaces** ⌈DiagnosticPortInterfaces shall by default interact with the machine local DM.⌋*()*
Note: It is recommend to use ara::com as communication binding between the ara::diag library and the DM process.
Note: Platform vendors should optionally support a diagnostic deployment over machine boundaries per process (i.e. the used communication binding should support communication across machines).

The AA could instantiated specialzed *DiagnosticPortInterfaces* for different purposes:

- DiagnosticRoutineInterface
  A typed interface for a single RoutineIdentifier
- DiagnosticRoutineGenericInterface
  A generic routine interface for multiple RoutineIdentifier(s)
- DiagnosticDataIdentifierInterface
  A typed data identifier interface for a single DataIdentifier
- DiagnosticDataIdentifierGenericInterface
  A generic data identifier interface for multiple DataIdentifier(s)
- DiagnosticDataElementInterface
  A typed data element interface for a single DataElement
- DiagnosticGenericUdsInterface
  A generic interface for diagnostic services
- DiagnosticMonitorInterface
  A interface for a single DiagnosticEvent

8.2.2 Monitor class

This interface is replacing the obsolete DiagnosticMonitor service interface. The constructor offers the possibility to add the debouncing options CounterBased or TimeBased. Further the functionality of the deprecated InitMonitorcould be added as a

notifier callback.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticMonitorInterface`.

[SWS_DM_00542]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::Monitor
Scope:	namespace ara::diag
Syntax:	`class Monitor final {...};`
Header file:	#include "ara/diag/monitor.h"
Description:	Class to implement operations on diagnostic Monitor interface.

⌋(*RS_Diag_04179*)

8.2.2.1 diag::Monitor::CounterBased

[SWS_DM_00538]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::Monitor::CounterBased
Scope:	class ara::diag::Monitor
Syntax:	`struct CounterBased {...};`
Header file:	#include "ara/diag/monitor.h"
Description:	Represents the parameters for counter-based debouncing.

⌋(*RS_Diag_04068*)

8.2.2.2 diag::Monitor::TimeBased

[SWS_DM_00539]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::Monitor::TimeBased
Scope:	class ara::diag::Monitor
Syntax:	`struct TimeBased {...};`
Header file:	#include "ara/diag/monitor.h"
Description:	Represents the parameters for time-based debouncing.

⌋(*RS_Diag_04225*)

8.2.2.3 diag::Monitor::InitMonitorReason

[SWS_DM_00540]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::Monitor::InitMonitorReason	
Scope:	class ara::diag::Monitor	
Underlying type:	–	
Syntax:	`enum class InitMonitorReason {...};`	
Values:	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.
Header file:	#include "ara/diag/monitor.h"	
Description:	Represents the status information reported to AAs why the monitor may be re-initalized.	

⌋(*RS_Diag_04179*)

8.2.2.4 diag::Monitor::MonitorAction

[SWS_DM_00541]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::Monitor::MonitorAction	
Scope:	class ara::diag::Monitor	
Underlying type:	–	
Syntax:	`enum class MonitorAction {...};`	
Values:	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 ExtendedData Records 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.
Header file:	#include "ara/diag/monitor.h"	
Description:	Represents the status information reported by AAs being relevant for error monitoring.	

⌋(*RS_Diag_04179*)

8.2.2.5 diag::Monitor::Monitor

[SWS_DM_00548]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Monitor::Monitor(const ara::core::InstanceSpecifier &specifier, std::Function< void(ara::diag::InitMonitorReason)> initMonitor, std::Function< std::sint8_t()> get_fault_ detection_counter)	
Scope:	class ara::diag::Monitor	
Syntax:	`Monitor (const ara::core::InstanceSpecifier &specifier, std::Function< void(ara::diag::InitMonitorReason)> initMonitor, std::Function< std::sint8_t()> get_fault_detection_counter);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticMonitorInterface
	initMonitor	Possibility to register an InitMonitor callback
	get_fault_detection_counter	Possibility to register a function to get the current FDC for this event.
Header file:	#include "ara/diag/monitor.h"	
Description:	Monitor constructor for Monitors with Monitor-internal debouncing.	

⌋(*RS_AP_00137*, *RS_Diag_04179*)

[SWS_DM_00549]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Monitor::Monitor(const ara::core::InstanceSpecifier &specifier, std::Function< void(ara::diag::InitMonitorReason)> initMonitor, CounterBased debouncing)	
Scope:	class ara::diag::Monitor	
Syntax:	`Monitor (const ara::core::InstanceSpecifier &specifier, std::Function< void(ara::diag::InitMonitorReason)> initMonitor, CounterBased debouncing);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticMonitorInterface
	initMonitor	Possibility to register an InitMonitor callback
	debouncing	CounterBased debouncing option is added to the monitor
Header file:	#include "ara/diag/monitor.h"	
Description:	Monitor constructor for Monitors with counter-based debouncing.	

⌋(*RS_AP_00137*, *RS_Diag_04179*, *RS_Diag_04068*)

[SWS_DM_00550]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::Monitor::Monitor(const ara::core::InstanceSpecifier &specifier, std::Function< void(ara::diag::InitMonitorReason)> initMonitor, TimeBased debouncing)
Scope:	class ara::diag::Monitor
Syntax:	`Monitor (const ara::core::InstanceSpecifier &specifier, std::Function< void(ara::diag::InitMonitorReason)> initMonitor, TimeBased debouncing);`

▽

△

Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticMonitorInterface
	initMonitor	Possibility to register an InitMonitor callback
	debouncing	TimeBased debouncing option is added to the monitor
Header file:	#include "ara/diag/monitor.h"	
Description:	Monitor constructor for Monitors with time-based debouncing.	

⌋(*RS_AP_00137*, *RS_Diag_04179*, *RS_Diag_04225*)

8.2.2.6 diag::Monitor::ReportMonitorAction

[SWS_DM_00543]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Monitor::ReportMonitorAction(ara::diag::MonitorAction action)	
Scope:	class ara::diag::Monitor	
Syntax:	`ara::core::Result<void> ReportMonitorAction (ara::diag::MonitorAction action);`	
Parameters (in):	action	Contains either the last (un-)qualified test result of the diagnostic monitor or commands to control the debouncing or to force a prestorage.
Return value:	ara::core::Result< void >	a Result with either void or an error
Errors:	tbd	This error includes errors in reporting.
Header file:	#include "ara/diag/monitor.h"	
Description:	Function to report the status information being relevant for error monitoring paths.	

⌋(*RS_Diag_04179*, *RS_AP_00139*)

8.2.3 GenericUDSService class

This interface allows a generic implementation to handle UDS messages. Several `DiagnosticServiceSwMapping`s with a reference to *tbd* can map to the same PortPrototype.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticGenericUdsInterface`.

[SWS_DM_00602]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::GenericUDSService
Scope:	namespace ara::diag
Syntax:	`class GenericUDSService {...};`
Header file:	#include "ara/diag/generic_uds_service.h"
Description:	Generic UDS interface.

⌋()

8.2.3.1 diag::GenericUDSService::OperationOutput

[SWS_DM_00578]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::GenericUDSService::OperationOutput
Scope:	class ara::diag::GenericUDSService
Syntax:	`struct OperationOutput {...};`
Header file:	#include "ara/diag/generic_uds_service.h"
Description:	Response data of positive respone message.

⌋()

8.2.3.2 diag::GenericUDSService::GenericUDSService function

[SWS_DM_00616]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericUDSService::GenericUDSService(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::GenericUDSService	
Syntax:	`explicit GenericUDSService (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	An InstanceSpecifier linking this instance with the PortPrototype in the manifest
Header file:	#include "ara/diag/generic_uds_service.h"	
Description:	Constructor of GenericUDSService.	

⌋(*RS_AP_00137*)

8.2.3.3 diag::GenericUDSService::~GenericUDSService function

[SWS_DM_00584]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::GenericUDSService::~GenericUDSService()
Scope:	class ara::diag::GenericUDSService
Syntax:	`virtual ~GenericUDSService () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/generic_uds_service.h"
Description:	Destructor of GenericUDSService.

⌋(*RS_AP_00134*)

8.2.3.4 diag::GenericUDSService::Offer function

[SWS_DM_00619]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericUDSService::Offer()	
Scope:	class ara::diag::GenericUDSService	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/generic_uds_service.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*)

8.2.3.5 diag::GenericUDSService::StopOffer function

[SWS_DM_00620]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::GenericUDSService::StopOffer()
Scope:	class ara::diag::GenericUDSService
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/generic_uds_service.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.2.3.6 diag::GenericUDSService::HandleMessage function

[SWS_DM_00618]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericUDSService::HandleMessage(std::uint8_t sid, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::GenericUDSService	
Syntax:	`virtual ara::core::Future<OperationOutput> HandleMessage (std::uint8_t sid, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	sid	Diagnostic Request Service Identifier.
	request_data	Diagnostic request data (starting after SID).
	meta_info	MetaInfo of the request.
	cancellation_handler	Set if the current conversation is canceled.
Return value:	ara::core::Future< OperationOutput >	a Result with either a OperationOutput (Diagnostic response data (starting after SID)) or an error
Errors:	tbd	This error set includes all NegativeResponseCodes defined in UDS.
Header file:	#include "ara/diag/generic_uds_service.h"	
Description:	Called for any request messsage.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.2.4 GenericDataIdentifier class

This interface allows a generic implementation of an data identifier handler. Multiple `DiagnosticServiceDataIdentifierPortMapping`s can reference to the same PortPrototype.
The InstanceSpecifier is only compatible with PortInterface of `Diagnostic-DataIdentifierGenericInterface`.

[SWS_DM_00607]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::GenericDataIdentifier
Scope:	namespace ara::diag
Syntax:	`class GenericDataIdentifier {...};`
Header file:	#include "ara/diag/generic_data_identifier.h"
Description:	Generic DataIdentifer interface.

⌋()

8.2.4.1 diag::GenericDataIdentifier::OperationOutput type

[SWS_DM_00641]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::GenericDataIdentifier::OperationOutput
Scope:	class ara::diag::GenericDataIdentifier
Syntax:	`struct OperationOutput {...};`
Header file:	#include "ara/diag/generic_data_identifier.h"
Description:	Response data of positive respone message.

⌋()

8.2.4.2 diag::GenericDataIdentifier::GenericDataIdentifier function

[SWS_DM_00634]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericDataIdentifier::GenericDataIdentifier(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::GenericDataIdentifier	
Syntax:	`explicit GenericDataIdentifier (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticDataIdentifierGenericInterface
Header file:	#include "ara/diag/generic_data_identifier.h"	
Description:	Class for an GenericDataIdentifier.	

⌋(*RS_AP_00137*)

8.2.4.3 diag::GenericDataIdentifier::~GenericDataIdentifier function

[SWS_DM_00635]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::GenericDataIdentifier::~GenericDataIdentifier()
Scope:	class ara::diag::GenericDataIdentifier
Syntax:	`virtual ~GenericDataIdentifier () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/generic_data_identifier.h"
Description:	Destructor of class GenericDataIdentifier.

⌋(*RS_AP_00134*)

8.2.4.4 diag::GenericDataIdentifier::Offer function

[SWS_DM_00638]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericDataIdentifier::Offer()	
Scope:	class ara::diag::GenericDataIdentifier	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/generic_data_identifier.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*)

8.2.4.5 diag::GenericDataIdentifier::StopOffer function

[SWS_DM_00639]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::GenericDataIdentifier::StopOffer()
Scope:	class ara::diag::GenericDataIdentifier
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/generic_data_identifier.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.2.4.6 diag::GenericDataIdentifier::Read function

[SWS_DM_00636]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericDataIdentifier::Read(std::uint16_t data_identifier, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::GenericDataIdentifier	
Syntax:	`virtual ara::core::Future<OperationOutput> Read (std::uint16_t data_identifier, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	data_identifier	the corresponding DataIdentifer
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)

▽

△

Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/generic_data_identifier.h"	
Description:	Called for ReadDataByIdentifer request for this DiagnosticDataIdentifier.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.2.4.7 diag::GenericDataIdentifier::Write function

[SWS_DM_00637]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericDataIdentifier::Write(std::uint16_t data_identifier, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::GenericDataIdentifier	
Syntax:	`virtual ara::core::Future<void> Write (std::uint16_t data_identifier, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	data_identifier	the corresponding DataIdentifer
	request_data	Content of request message (without DataIdentifer)
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< void >	a Result with either void (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/generic_data_identifier.h"	
Description:	Called for WriteDataByIdentifer request for this DiagnosticDataIdentifier.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.2.5 GenericRoutine class

This interface allows a generic implementation of an routine handler. Several `DiagnosticServiceSwMapping`s with a reference to `DiagnosticRoutineControl` can map to the same PortPrototype.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticRoutineGenericInterface`.

[SWS_DM_00605]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::GenericRoutine
Scope:	namespace ara::diag
Syntax:	`class GenericRoutine {...};`
Header file:	#include "ara/diag/generic_routine.h"
Description:	Generic Routine interface.

⌋(*RS_Diag_04224*)

8.2.5.1 diag::GenericRoutine::OperationOutput

[SWS_DM_00551]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::GenericRoutine::OperationOutput
Scope:	class ara::diag::GenericRoutine
Syntax:	`struct OperationOutput {...};`
Header file:	#include "ara/diag/generic_routine.h"
Description:	Response data of positive respone message.

⌋(*RS_Diag_04224*)

8.2.5.2 diag::GenericRoutine::GenericRoutine function

[SWS_DM_00552]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericRoutine::GenericRoutine(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::GenericRoutine	
Syntax:	`explicit GenericRoutine (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticRoutineGenericInterface
Header file:	#include "ara/diag/generic_routine.h"	
Description:	Class for an GenericRoutine.	

⌋(*RS_AP_00137*, *RS_Diag_04224*)

8.2.5.3 diag::GenericRoutine::~GenericRoutine function

[SWS_DM_00553]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::GenericRoutine::~GenericRoutine()
Scope:	class ara::diag::GenericRoutine
Syntax:	`virtual ~GenericRoutine () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/generic_routine.h"
Description:	Destructor of class GenericRoutine.

⌋(*RS_AP_00134*, *RS_Diag_04224*)

8.2.5.4 diag::GenericRoutine::Offer function

[SWS_DM_00557]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericRoutine::Offer()	
Scope:	class ara::diag::GenericRoutine	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/generic_routine.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00138*, *RS_Diag_04224*)

8.2.5.5 diag::GenericRoutine::StopOffer function

[SWS_DM_00558]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::GenericRoutine::StopOffer()
Scope:	class ara::diag::GenericRoutine
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/generic_routine.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.2.5.6 diag::GenericRoutine::Start function

[SWS_DM_00554]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericRoutine::Start(std::uint16_t routine_id, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::GenericRoutine	
Syntax:	`virtual ara::core::Future<OperationOutput> Start (std::uint16_t routine_id, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	routine_id	the corresponding RoutineIdentifer
	request_data	Content of request message (without Routine Identifer)
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/generic_routine.h"	
Description:	Called for RoutineControl with SubFunction Start request for this DiagnosticRoutineIdentifier.	

⌋(*RS_AP_00138*, *RS_Diag_04224*, *RS_Diag_04170*)

8.2.5.7 diag::GenericRoutine::Stop function

[SWS_DM_00555]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericRoutine::Stop(std::uint16_t routine_id, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::GenericRoutine	
Syntax:	`virtual ara::core::Future<OperationOutput> Stop (std::uint16_t routine_id, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	routine_id	the corresponding RoutineIdentifer
	request_data	Content of request message (without Routine Identifer)
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/generic_routine.h"	
Description:	Called for RoutineControl with SubFunction Stop request for this DiagnosticRoutineIdentifier.	

⌋(*RS_AP_00138*, *RS_Diag_04224*, *RS_Diag_04170*)

8.2.5.8 diag::GenericRoutine::RequestResults function

[SWS_DM_00556]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::GenericRoutine::RequestResults(std::uint16_t routine_id, ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::GenericRoutine	
Syntax:	`virtual ara::core::Future<OperationOutput> RequestResults (std::uint16_t routine_id, ara::core::Span< std::uint8_t > request_ data, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	routine_id	the corresponding RoutineIdentifer
	request_data	Content of request message (without Routine Identifer)
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/generic_routine.h"	
Description:	Called for RoutineControl with SubFunction RequestResults request for this DiagnosticRoutine Identifier.	

⌋(*RS_Diag_04224*, *RS_Diag_04170*)

8.2.6 CancellationHandler class

[SWS_DM_00608]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::CancellationHandler
Scope:	namespace ara::diag
Syntax:	`class CancellationHandler final {...};`
Header file:	#include "ara/diag/cancellation_handler.h"
Description:	CancellationHandler contains a shared state if the processing should be canceled .

⌋()

8.2.6.1 diag::CancellationHandler::CancellationHandler function

[SWS_DM_00609]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::CancellationHandler::CancellationHandler()
Scope:	class ara::diag::CancellationHandler
Syntax:	`CancellationHandler ()=delete;`
Header file:	#include "ara/diag/cancellation_handler.h"
Description:	Constructor of CancellationHandler cannot be used.

⌋()

[SWS_DM_00610]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::CancellationHandler::CancellationHandler(CancellationHandler &&)	
Scope:	class ara::diag::CancellationHandler	
Syntax:	`CancellationHandler (CancellationHandler &&) noexcept=default;`	
DIRECTION NOT DEFINED	CancellationHandler &&	–
Exception Safety:	noexcept	
Header file:	#include "ara/diag/cancellation_handler.h"	
Description:	Move constructor of CancellationHandler.	

⌋()

[SWS_DM_00611]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::CancellationHandler::CancellationHandler(CancellationHandler &)
Scope:	class ara::diag::CancellationHandler
Syntax:	`CancellationHandler (CancellationHandler &)=delete;`
Header file:	#include "ara/diag/cancellation_handler.h"
Description:	Copy constructor of CancellationHandler cannot be used.

⌋()

[SWS_DM_00612]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::CancellationHandler::operator=(CancellationHandler &&)	
Scope:	class ara::diag::CancellationHandler	
Syntax:	`CancellationHandler& operator= (CancellationHandler &&) noexcept=default;`	
DIRECTION NOT DEFINED	CancellationHandler &&	–
Return value:	CancellationHandler &	–
Exception Safety:	noexcept	

▽

△

Header file:	#include "ara/diag/cancellation_handler.h"
Description:	Move assignment operator of CancellationHandler.

⌋()

[SWS_DM_00613]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::CancellationHandler::operator=(CancellationHandler &)
Scope:	class ara::diag::CancellationHandler
Syntax:	`CancellationHandler& operator= (CancellationHandler &)=delete;`
Header file:	#include "ara/diag/cancellation_handler.h"
Description:	Copy assignment operator of CancellationHandler cannot be used.

⌋()

8.2.6.2 diag::CancellationHandler::IsCanceled function

[SWS_DM_00614]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::CancellationHandler::IsCanceled()	
Scope:	class ara::diag::CancellationHandler	
Syntax:	`bool IsCanceled () const;`	
Return value:	bool	–
Header file:	#include "ara/diag/cancellation_handler.h"	
Description:	Returns true in if the diagnostic service execution is cancelled in DM.	

⌋()

8.2.6.3 diag::CancellationHandler::SetNotifier function

[SWS_DM_00615]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::CancellationHandler::SetNotifier(std::Function< void()> notifier)	
Scope:	class ara::diag::CancellationHandler	
Syntax:	`void SetNotifier (std::Function< void()> notifier);`	
DIRECTION NOT DEFINED	notifier	–

▽

△

Return value:	None
Header file:	#include "ara/diag/cancellation_handler.h"
Description:	Regisering a notifier function which is called if the diagnostic service execution is canceled in DM.

⌋()

8.3 C++ Diagnostic generated API Interfaces

Namespaces are used to separate the definition of services from each other to prevent name conflicts and they allow to use reasonably short names.

[SWS_DM_00510]{DRAFT} **Namespace of Service header files** ⌈Based on the `symbol` attributes of the ordered `SymbolProps` aggregated by `PortInterface` in role `namespace`, the C++ namespace of the *Service header file* shall be:

```
namespace <PortInterface.namespace[0].symbol> {
namespace <PortInterface.namespace[1].symbol> {
namespace <...> {
namespace <PortInterface.namespace[n].symbol> {
...
} // namespace <PortInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <PortInterface.namespace[1].symbol>
} // namespace <PortInterface.namespace[0].symbol>
```

with all namespace names converted to lower-case letters.⌋()

8.3.1 Implementation Types header files

The *Implementation Types header files* include the ara::diag specific type declarations derived from the `CppImplementationDataType`s created from the definitions of AUTOSAR meta model classes within the `DiagnosticPortInterface` description.

[SWS_DM_00511]{DRAFT} **Implementation Types header files existence** ⌈The diagnostic management shall provide an *Implementation Types header file* for each `CppImplementationDataType` defined in the input by using the file name `impl_type_<symbol>.h`, where `<symbol>` is the *Cpp Implementation Data Type symbol* converted to lower-case letters.⌋()

The *Implementation Types header files* might need to include other header files, e.g. for `ara::core::String` or `ara::core::Vector`.

[SWS_DM_00512]{DRAFT} **Data Type definitions for AUTOSAR Data Types in Implementation Types header files** ⌈The *Implementation Types header files* shall include the type definitions and structure and class definitions for all the AUTOSAR Data Types.⌋()

[SWS_DM_00513]{DRAFT} **Implementation Types header file namespace** ⌈The C++ namespace of the *Implementation Types header file* for a given CppImplementationDataType is defined via the aggregated namespace. Based on the symbol attributes of the ordered SymbolProps aggregated by CppImplementationDataType in role namespace, the C++ namespace of the *Implementation Types header file* shall be:

```
namespace <CppImplementationDataType.namespace[0].symbol> {
namespace <CppImplementationDataType.namespace[1].symbol> {
namespace <...> {
namespace <CppImplementationDataType.namespace[n].symbol> {
...
} // namespace <CppImplementationDataType.namespace[n].symbol>
} // namespace <...>
} // namespace <CppImplementationDataType.namespace[1].symbol>
} // namespace <CppImplementationDataType.namespace[0].symbol>
```

with all namespace names converted to lower-case letters.⌋()

8.3.2 Typed Routine class

This routine interface is replacing the obsolete RoutineService service interface. The InstanceSpecifier is only compatible with PortInterface of DiagnosticRoutineInterface.

[SWS_DM_00604]{DRAFT} ⌈

Kind:	class
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_PortInterface
Scope:	namespace Namespace_1_OfPortInterface::Namespace_2_OfPortInterface
Syntax:	`class ShortnameOf_RI_PortInterface {...};`
Header file:	#include "ara/diag/name_routine.h" #include "ara/diag/Namespace_1_OfPortInterface/Namespace_2_OfPortInterface/.../ShortnameOf_PortInterface_routine.h"
Description:	Typed Routine interface.

⌋()

8.3.2.1 diag::Routine::StartOutput

[SWS_DM_00581]{DRAFT} ⌈

Kind:	struct
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::StartOutput
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface
Syntax:	`struct StartOutput {...};`
Header file:	#include "ara/diag/name_routine.h"
Description:	Response data.

⌋()

8.3.2.2 diag::Routine::StopOutput

[SWS_DM_00582]{DRAFT} ⌈

Kind:	struct
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::StopOutput
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface
Syntax:	`struct StopOutput {...};`
Header file:	#include "ara/diag/name_routine.h"
Description:	Response data.

⌋()

8.3.2.3 diag::Routine::RequestResultsOutput

[SWS_DM_00583]{DRAFT} ⌈

Kind:	struct
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::RequestResultsOutput
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface
Syntax:	`struct RequestResultsOutput {...};`
Header file:	#include "ara/diag/name_routine.h"
Description:	Response data.

⌋()

8.3.2.4 Routine Constructor function

[SWS_DM_00589]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::ShortnameOfPortInterface(const ara::core::InstanceSpecifier &specifier)	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface	
Syntax:	`explicit ShortnameOfPortInterface (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	An InstanceSpecifier linking this instance with the PortPrototype in the manifest
Header file:	#include "ara/diag/name_routine.h"	
Description:	Constructor of typed Routine interface.	

⌋(*RS_AP_00137*)

8.3.2.5 Routine Destructor function

[SWS_DM_00590]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::~ShortnameOfPortInterface()
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface
Syntax:	`virtual ~ShortnameOfPortInterface () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/name_routine.h"
Description:	Destructor of typed Routine interface.

⌋(*RS_AP_00134*)

8.3.2.6 Routine ::Offer function

[SWS_DM_00594]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::Offer()	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/name_routine.h"	

▽

△

Description:	This Offer will enable the DM to forward request messages to this handler.

⌋(*RS_Diag_04224*, *RS_AP_00139*)

8.3.2.7 Routine ::StopOffer function

[SWS_DM_00595]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::StopOffer()
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/name_routine.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.3.2.8 Routine::Start function

[SWS_DM_00591]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::Start(Namespace_1_OfTypeOfArgumentDataPrototype::Type_1_OfArgumentData Prototype Shortname_1_OfArgumentDataPrototype;Namespace_2_OfTypeOfArgumentData Prototype::Type_2_OfArgumentDataPrototype Shortname_2_OfArgumentDataPrototype;... ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface	
Syntax:	`virtual ara::core::Future<StartOutput> Start (Namespace_1_OfTypeOf ArgumentDataPrototype::Type_1_OfArgumentDataPrototype Shortname_1_Of ArgumentDataPrototype;Namespace_2_OfTypeOfArgumentData Prototype::Type_2_OfArgumentDataPrototype Shortname_2_OfArgumentData Prototype;... ara::diag::MetaInfo meta_info, ara::diag::Cancellation Handler cancellation_handler)=0;`	
Parameters (in):	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< StartOutput >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue

▽

△

Header file:	#include "ara/diag/name_routine.h"
Description:	Called for RoutineControl with SubFunction Start request for this DiagnosticRoutineIdentifier.

⌋(*RS_AP_00138*, *RS_Diag_04224*, *RS_Diag_04170*)

8.3.2.9 Routine::Stop function

[SWS_DM_00592]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::Stop(Namespace_1_OfTypeOfArgumentDataPrototype::Type_1_OfArgumentData Prototype Shortname_1_OfArgumentDataPrototype;Namespace_2_OfTypeOfArgumentData Prototype::Type_2_OfArgumentDataPrototype Shortname_2_OfArgumentDataPrototype;... ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface	
Syntax:	`virtual ara::core::Future<StopOutput> Stop (Namespace_1_OfTypeOf ArgumentDataPrototype::Type_1_OfArgumentDataPrototype Shortname_1_Of ArgumentDataPrototype;Namespace_2_OfTypeOfArgumentData Prototype::Type_2_OfArgumentDataPrototype Shortname_2_OfArgumentData Prototype;... ara::diag::MetaInfo meta_info, ara::diag::Cancellation Handler cancellation_handler)=0;`	
Parameters (in):	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< StopOutput >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/name_routine.h"	
Description:	Called for RoutineControl with SubFunction Stop request for this DiagnosticRoutineIdentifier.	

⌋(*RS_AP_00138*, *RS_Diag_04224*, *RS_Diag_04170*)

8.3.2.10 Routine::RequestResults function

[SWS_DM_00593]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface::RequestResults(Namespace_1_OfTypeOfArgumentDataPrototype::Type_1_Of ArgumentDataPrototype Shortname_1_OfArgumentDataPrototype;Namespace_2_OfTypeOf ArgumentDataPrototype::Type_2_OfArgumentDataPrototype Shortname_2_OfArgumentData Prototype;... ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_ handler)

▽

△

Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_RI_Port Interface	
Syntax:	`virtual ara::core::Future<RequestResultsOutput> RequestResults (Namespace_1_OfTypeOfArgumentDataPrototype::Type_1_OfArgumentData Prototype Shortname_1_OfArgumentDataPrototype;Namespace_2_OfTypeOf ArgumentDataPrototype::Type_2_OfArgumentDataPrototype Shortname_2_Of ArgumentDataPrototype;... ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< RequestResults Output >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/name_routine.h"	
Description:	Called for RoutineControl with SubFunction RequestResults request for this DiagnosticRoutine Identifier.	

⌋(*RS_AP_00138*, *RS_Diag_04224*, *RS_Diag_04170*)

8.3.3 Typed DataIdentifier class

This data identifier interface is replacing the obsolete `DataIdentifier` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticDataIdentifierInterface`.

[SWS_DM_00601]{DRAFT} ⌈

Kind:	class
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface
Scope:	namespace Namespace_1_OfPortInterface::Namespace_2_OfPortInterface
Syntax:	`class ShortnameOf_DI_PortInterface {...};`
Header file:	#include "ara/diag/Namespace_1_OfPortInterface/Namespace_2_OfPortInterface/.../ ShortnameOf_PortInterface_data_identifier.h"
Description:	Typed DataIdentifer interface.

⌋()

8.3.3.1 diag::DataIdentifier::OperationOutput

[SWS_DM_00579]{DRAFT} ⌈

Kind:	struct
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface::Output
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface
Syntax:	`struct Output {...};`
Header file:	#include "ara/diag/name_data_identifier.h"
Description:	Response data.

⌋()

8.3.3.2 DataIdentifier Constructor function

[SWS_DM_00585]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface::ShortnameOfPortInterface(const ara::core::InstanceSpecifier &specifier)	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface	
Syntax:	`explicit ShortnameOfPortInterface (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	An InstanceSpecifier linking this instance with the PortPrototype in the manifest
Header file:	#include "ara/diag/name_data_identifier.h"	
Description:	Constructor of typed DataIdentifer interface.	

⌋(*RS_AP_00137*)

8.3.3.3 DataIdentifier Destructor function

[SWS_DM_00586]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface::~ShortnameOfPortInterface()
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface
Syntax:	`virtual ~ShortnameOfPortInterface () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/name_data_identifier.h"
Description:	Destructor of typed DataIdentifer interface.

⌋(*RS_AP_00134*)

8.3.3.4 DataIdentifier ::Offer function

[SWS_DM_00599]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface::Offer()	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/name_data_identifier.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*)

8.3.3.5 DataIdentifier ::StopOffer function

[SWS_DM_00600]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface::StopOffer()
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/name_data_identifier.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.3.3.6 DataIdentifier::Read function

[SWS_DM_00640]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface::Read(ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_ handler)
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface
Syntax:	`virtual ara::core::Future<Output> Read (ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`

▽

△

Parameters (in):	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< Output >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/name_data_identifier.h"	
Description:	Called for ReadDataByIdentifer request for this DiagnosticDataIdentifier.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.3.3.7 DataIdentifier::Write function

[SWS_DM_00598]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface::Write(Namespace_1_OfTypeOfArgumentDataPrototype::Type_1_OfArgumentData Prototype Shortname_1_OfArgumentDataPrototype;Namespace_2_OfTypeOfArgumentData Prototype::Type_2_OfArgumentDataPrototype Shortname_2_OfArgumentDataPrototype;... ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DI_Port Interface	
Syntax:	`virtual ara::core::Future<void> Write (Namespace_1_OfTypeOfArgument DataPrototype::Type_1_OfArgumentDataPrototype Shortname_1_OfArgument DataPrototype;Namespace_2_OfTypeOfArgumentDataPrototype::Type_2_Of ArgumentDataPrototype Shortname_2_OfArgumentDataPrototype;... ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< void >	a Result with either void (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/name_data_identifier.h"	
Description:	Called for WriteDataByIdentifer request for this DiagnosticDataIdentifier.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.3.4 Typed DataElement class

This data element interface is replacing the obsolete `DataElement` service interface. The InstanceSpecifier is only compatible with PortInterface of `DiagnosticDataElementInterface`.

[SWS_DM_00603]{DRAFT} ⌈

Kind:	class
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface
Scope:	namespace Namespace_1_OfPortInterface::Namespace_2_OfPortInterface
Syntax:	`class ShortnameOf_DE_PortInterface {...};`
Header file:	#include "ara/diag/Namespace_1_OfPortInterface/Namespace_2_OfPortInterface/.../ ShortnameOf_PortInterface_data_element.h"
Description:	Typed DataElement interface.

⌋()

8.3.4.1 diag::DataElement::OperationOutput

[SWS_DM_00580]{DRAFT} ⌈

Kind:	struct
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface::OperationOutput
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface
Syntax:	`struct OperationOutput {...};`
Header file:	#include "ara/diag/name_data_element.h"
Description:	Response data.

⌋()

8.3.4.2 DataElement Constructor function

[SWS_DM_00587]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface::ShortnameOfPortInterface(const ara::core::InstanceSpecifier &specifier)	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface	
Syntax:	`explicit ShortnameOfPortInterface (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	An InstanceSpecifier linking this instance with the PortPrototype in the manifest
Header file:	#include "ara/diag/name_data_element.h"	
Description:	Constructor of typed DataElement interface.	

⌋(*RS_AP_00137*)

8.3.4.3 DataElement Destructor function

[SWS_DM_00588]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface::~ShortnameOfPortInterface()
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface
Syntax:	`virtual ~ShortnameOfPortInterface () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/name_data_element.h"
Description:	Destructor of typed DataElement interface.

⌋(*RS_AP_00134*)

8.3.4.4 DataElement ::Offer function

[SWS_DM_00597]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface::Offer()	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/name_data_element.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*)

8.3.4.5 DataElement ::StopOffer function

[SWS_DM_00617]{DRAFT} ⌈

Kind:	function
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface::StopOffer()
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface
Syntax:	`void StopOffer ();`

▽

△

Return value:	None
Header file:	#include "ara/diag/name_data_element.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.3.4.6 DataElement ::Read function

[SWS_DM_00596]{DRAFT} ⌈

Kind:	function	
Symbol:	Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface::Read(ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_ handler)	
Scope:	class Namespace_1_OfPortInterface::Namespace_2_OfPortInterface::ShortnameOf_DE_Port Interface	
Syntax:	`virtual ara::core::Future<OperationOutput> Read (ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Result with either OperationOutput (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/name_data_element.h"	
Description:	Called for reading a DataElement.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.4 C++ Diagnostic Error Types

[SWS_DM_00544]{DRAFT} **Use of general ara::diag errors** ⌈Any `Checked Error` of a service interface shall be reported via the return type as specified in [14].⌋()

In `ara::diag`, there are the following types of Checked Errors:

1. Offer ara::diag errors: These errors can occur in a call of a any Offer interface method. They are defined in the error domain `ara::diag::DiagErrorDomain`.
2. Reporting ara::diag errors: These errors can occur in a call of a ReportMonitorAction interface method. They are defined in the error domain `ara::diag::DiagErrorDomain`.
3. UDS NRC ara::diag errors: These errors can be returned by the skeletons. They are defined in the error domain `ara::diag::DiagUdsNrcErrorDomain`.

[SWS_DM_00545]{DRAFT} **Definition Offer ara::diag errors** ⌈Offer ara::diag errors shall be defined in the error domain `ara::diag::DiagErrorDomain` in accordance with [14].⌋*()*

[SWS_DM_00559]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::DiagOfferErrc	
Scope:	namespace ara::diag	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class DiagOfferErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kAlreadyOffered= 101	The service is already offered.
	kConfigurationMismatch= 102	monitor configuration does not match dext
	kDebouncingConfiguration Inconsistent= 103	monitor debouncing configuration invalid, e.g. passed threshold larger than failed threshold...
Header file:	#include "ara/diag/diag_error_domain.h"	
Description:	The DiagOfferErrc enumeration defines the error codes for the DiagErrorDomain.	

⌋*()*

[SWS_DM_00546]{DRAFT} **Definition Reporting ara::diag errors** ⌈Reporting ara::diag errors shall be defined in the error domain `ara::diag::DiagErrorDomain` in accordance with [14].⌋*()*

[SWS_DM_00560]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::DiagReportingErrc	
Scope:	namespace ara::diag	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class DiagReportingErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kInvalidArgument= 105	e.g. kPreFailed with internal debouncing
	kGenericError= 107	generic issue, e.g. connection to DM lost
Header file:	#include "ara/diag/diag_error_domain.h"	
Description:	The DiagOfferErrc enumeration defines the error codes for the DiagErrorDomain. .	

⌋*()*

[SWS_DM_00547]{DRAFT} **Definition UDS NRC ara::diag errors** ⌈UDS NRC ara::diag errors shall be defined in the error domain `ara::diag::DiagUdsNrcErrorDomain` in accordance with [14].⌋*()*

[SWS_DM_00526]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::DiagUdsNrcErrc	
Scope:	namespace ara::diag	
Underlying type:	int32_t	
Syntax:	`enum class DiagUdsNrcErrc : int32_t {...};`	
Values:	kGeneralReject= 0x10	According to ISO.
	kServiceNotSupported= 0x11	According to ISO.
	kSubfunctionNotSupported= 0x12	According to ISO.
	kIncorrectMessageLengthOrInvalid Format= 0x13	According to ISO.
	kResponseTooLong= 0x14	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.
	kRequiredTimeDelayNotExpired= 0x37	According to ISO.
	kUploadDownloadNotAccepted= 0x70	According to ISO.
	kTransferDataSuspended= 0x71	According to ISO.
	kGeneralProgrammingFailure= 0x72	According to ISO.
	kWrongBlockSequenceCounter= 0x73	According to ISO.
	kSubFunctionNotSupportedInActive Session= 0x7E	According to ISO.
	kServiceNotSupportedInActive Session= 0x7F	According to ISO.
	kRpmTooHigh= 0x81	According to ISO.
	kRpmTooLow= 0x82	According to ISO.
	kEngineIsRunning= 0x83	According to ISO.
	kEngineIsNotRunning= 0x84	According to ISO.
	kEngineRunTimeTooLow= 0x85	According to ISO.
	kTemperatureTooHigh= 0x86	According to ISO.
	kTemperatureTooLow= 0x87	According to ISO.
	kVehicleSpeedTooHigh= 0x88	According to ISO.
	kVehicleSpeedTooLow= 0x89	According to ISO.
	kThrottlePedalTooHigh= 0x8A	According to ISO.
	kThrottlePedalTooLow= 0x8B	According to ISO.
	kTransmissionRangeNotInNeutral= 0x8C	According to ISO.
	kTransmissionRangeNotInGear= 0x8D	According to ISO.

△

	kBrakeSwitchNotClosed= 0x8F	According to ISO.
	kShifterLeverNotInPark= 0x90	According to ISO.
	kTorqueConverterClutchLocked= 0x91	According to ISO.
	kVoltageTooHigh= 0x92	According to ISO.
	kVoltageTooLow= 0x93	According to ISO.
	kNoProcessingNoResponse= 0xFF	Deviating from ISO - no further service processing and no response (silently ignore request message).
Header file:	#include "ara/diag/diag_uds_nrc_error_domain.h"	
Description:	Specifies the types of internal errors that can occur upon calling Offer or ReportMonitorAction.	

⌋()

8.5 C++ Diagnostic API Interfaces

This chapter is considered to be experimental and thus might be subject to design changes and additional interfaces in the upcoming release. This chapter lists all experimental C++ API interfaces of the DM for interaction with application.

service interface	*diagnostic interface*
DiagnosticConversation	-
DiagnosticEvent	ara::diag::Event
DTCInformation DiagnosticMemory DiagnosticServer	ara::diag:DTCInformation
EnableCondition ClearCondition	ara::diag::Condition
OperationCycle	ara::diag::OperationCycle
Indicator	ara::diag::Indicator
ServiceManufacturerValidation ServiceSupplierValidation	ara::diag::ServiceValidation
SecurityAccess	ara::diag::SecurityAccess
DoIPGroupIdentification	ara::diag::DoIPGroupIdentification
DoIPPowerModeInformation	ara::diag::DoIPPowerMode

Table 8.1: Overview obsolete service interfaces with new C++ interfaces

8.5.1 Event class

This interface is replacing the obsolete `DiagnosticEvent` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticEventInterface`.

[SWS_DM_00646]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::Event
Scope:	namespace ara::diag
Syntax:	`class Event {...};`
Header file:	#include "ara/diag/event.h"
Description:	Class to implement operations on diagnostic Events.

⌋(*RS_Diag_04151*)

8.5.1.1 diag::Event::DTCFormatType type

[SWS_DM_00642]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::Event::DTCFormatType	
Scope:	class ara::diag::Event	
Underlying type:	uint8_t	
Syntax:	`enum class DTCFormatType : uint8_t {...};`	
Values:	kDTCFormatOBD= 0	SAE_J2012-DA_DTCFormat_00 as defined in ISO 15031-6 specification.
	kDTCFormatUDS= 1	ISO_14229-1_DTCFormat as defined in ISO 14229-1 specification.
	kDTCFormatJ1939= 2	SAE_J1939-73_DTCFormat as defined in SAE J1939-73.
Header file:	#include "ara/diag/event.h"	
Description:	Represents the type of the DTC format according to ISO 14229-1.	

⌋(*RS_Diag_04201*, *RS_AP_00125*)

8.5.1.2 diag::Event::EventStatusBit type

[SWS_DM_00643]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::Event::EventStatusBit	
Scope:	class ara::diag::Event	
Underlying type:	uint8_t	
Syntax:	`enum class EventStatusBit : uint8_t {...};`	
Values:	kTestFailed	bit 0: TestFailed
	kTestFailedThisOperationCycle	bit 1: TestFailedThisOperationCycle
	kTestNotCompletedThisOperation Cycle	bit 6: TestNotCompletedThisOperationCycle

▽

△

Header file:	#include "ara/diag/event.h"
Description:	Single event status bits.

⌋(*RS_Diag_04151*, *RS_AP_00125*)

8.5.1.3 diag::Event::EventStatusByte type

[SWS_DM_00644]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::Event::EventStatusByte
Scope:	class ara::diag::Event
Syntax:	`struct EventStatusByte : public uint8_t {...};`
Header file:	#include "ara/diag/event.h"
Description:	Current event status byte, bit-encoded.

⌋(*RS_Diag_04151*)

8.5.1.4 diag::Event::DebouncingState type

[SWS_DM_00645]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::Event::DebouncingState	
Scope:	class ara::diag::Event	
Underlying type:	uint8_t	
Syntax:	`enum class DebouncingState : uint8_t {...};`	
Values:	kNeutral= 0x00	Neutral (corresponds to FDC = 0)
	kTemporarilyDefective= 0x01	Temporarily Defective (corresponds to 0 < FDC < 127)
	kFinallyDefective= 0x02	finally Defective (corresponds to FDC = 127)
	kTemporarilyHealed= 0x04	temporarily healed (corresponds to -128 < FDC < 0)
	kFinallyHealed= 0x08	finally healed (corresponds to FDC = -128)
Header file:	#include "ara/diag/event.h"	
Description:	Debounce status of event .	

⌋(*RS_Diag_04068*, *RS_Diag_04225*, *RS_AP_00125*)

8.5.1.5 diag::Event::Event function

[SWS_DM_00647]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::Event(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::Event	
Syntax:	`explicit Event (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticEventInterface
Header file:	#include "ara/diag/event.h"	
Description:	Constructor fct. for objects of class Event.	

⌋(*RS_Diag_04151*, *RS_AP_00137*)

8.5.1.6 diag::Event::~Event function

[SWS_DM_00648]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::Event::~Event()
Scope:	class ara::diag::Event
Syntax:	`~Event () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/event.h"
Description:	Destructor of class Event.

⌋(*RS_Diag_04151*, *RS_AP_00134*)

8.5.1.7 diag::Event::GetEventStatus function

[SWS_DM_00649]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::GetEventStatus()	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<EventStatusByte> GetEventStatus ();`	
Return value:	ara::core::Result< EventStatusByte >	the current diagnostic event status
Header file:	#include "ara/diag/event.h"	
Description:	Returns the current diagnostic event status.	

⌋(*RS_Diag_04151*, *RS_AP_00139*)

8.5.1.8 diag::Event::SetEventStatusChangedNotifier function

[SWS_DM_00650]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::SetEventStatusChangedNotifier(std::Function< void(ara::diag::EventStatus Byte)> notifier)	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<void> SetEventStatusChangedNotifier (std::Function< void(ara::diag::EventStatusByte)> notifier);`	
Parameters (in):	notifier	The function to be called if a diagnostic event is changed.
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/event.h"	
Description:	Register a notifier function which is called if a diagnostic event is changed.	

⌋(*RS_Diag_04183*, *RS_AP_00139*)

8.5.1.9 diag::Event::GetLatchedWIRStatus function

[SWS_DM_00651]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::GetLatchedWIRStatus()	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<std::bool> GetLatchedWIRStatus ();`	
Return value:	ara::core::Result< std::bool >	the current warning indicator status
Header file:	#include "ara/diag/event.h"	
Description:	Returns the current warning indicator status.	

⌋(*RS_Diag_04204*, *RS_AP_00139*)

8.5.1.10 diag::Event::SetLatchedWIRStatus function

[SWS_DM_00652]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::SetLatchedWIRStatus(std::bool status)	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<void> SetLatchedWIRStatus (std::bool status);`	
Parameters (in):	status	Limp-home status as determined by the AA. '0' means limp-home not actice; '1' means limp-home actice;
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/event.h"	
Description:	Set the warning indicator status.	

⌋(*RS_Diag_04151*, *RS_AP_00139*)

8.5.1.11 diag::Event::GetDTCNumber function

[SWS_DM_00653]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::GetDTCNumber(ara::diag::DTCFormatType dtc_format)	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<std::uint32_t> GetDTCNumber (ara::diag::DTCFormatType dtc_format);`	
Parameters (in):	dtc_format	Define DTC format for the return value.
Return value:	ara::core::Result< std::uint32_t >	DTC number in respective DTCFormatType
Errors:	kNoSuchDTC	No DTC available.
Header file:	#include "ara/diag/event.h"	
Description:	Returns the DTC-ID related to this event instance.	

⌋(*RS_Diag_04201*, *RS_AP_00139*)

8.5.1.12 diag::Event::GetDebouncingStatus function

[SWS_DM_00654]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::GetDebouncingStatus()	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<ara::diag::DebouncingState> GetDebouncingStatus ();`	
Return value:	ara::core::Result< ara::diag::DebouncingState >	Return the current debouncing state of this event.
Header file:	#include "ara/diag/event.h"	
Description:	Get the current debouncing status .	

⌋(*RS_Diag_04068*, *RS_Diag_04225*, *RS_AP_00139*)

8.5.1.13 diag::Event::GetTestComplete function

[SWS_DM_00655]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::GetTestComplete()	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<std::bool> GetTestComplete ();`	
Return value:	ara::core::Result< std::bool >	Return the current test_completed-state of this event. "true", if FDC = -128 or FDC = 127; "false" in all other cases.

▽

△

Header file:	#include "ara/diag/event.h"
Description:	Get the status if the event has matured to test completed (corresponds to FDC = -128 or FDC = 127).

⌋(*RS_Diag_04151*, *RS_AP_00139*)

8.5.1.14 diag::Event::GetFaultDetectionCounter function

[SWS_DM_00656]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Event::GetFaultDetectionCounter()	
Scope:	class ara::diag::Event	
Syntax:	`ara::core::Result<std::sint8_t> GetFaultDetectionCounter ();`	
Return value:	ara::core::Result< std::sint8_t >	current FaultDetectionCounter value.
Header file:	#include "ara/diag/event.h"	
Description:	Returns the current value of Fault Detection Counter of this event.	

⌋(*RS_Diag_04068*, *RS_AP_00139*)

8.5.2 DTCInformation class

This interface is replacing the obsolete `DTCInformation`, `DiagnosticMemory` and `DiagnosticServer` service interfaces.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticDTCInformationInterface`.

[SWS_DM_00657]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::DTCInformation
Scope:	namespace ara::diag
Syntax:	`class DTCInformation {...};`
Header file:	#include "ara/diag/dtc_information.h"
Description:	Class to implement operations on DTC informations per configured DiagnosticMemory Destination.

⌋(*RS_Diag_04150*, *RS_Diag_04164*, *RS_Diag_04105*)

8.5.2.1 diag::DTCInformation::ControlDtcStatusType type

[SWS_DM_00663]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::DTCInformation::ControlDtcStatusType	
Scope:	class ara::diag::DTCInformation	
Underlying type:	–	
Syntax:	`enum class ControlDtcStatusType {...};`	
Values:	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.
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Type for ControlDTCStatus status as requested by UDS service 0x85 ControlDTCSetting.	

⌋(*RS_Diag_04159*)

8.5.2.2 diag::DTCInformation::UdsDtcStatusBitType type

[SWS_DM_00658]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::DTCInformation::UdsDtcStatusBitType	
Scope:	class ara::diag::DTCInformation	
Underlying type:	uint8_t	
Syntax:	`enum class UdsDtcStatusBitType : uint8_t {...};`	
Values:	kTestFailed= 0x01	bit 0: TestFailed
	kTestFailedThisOperationCycle= 0x02	bit 1: TestFailedThisOperationCycle
	kPendingDTC= 0x04	bit 2: PendingDTC
	kConfirmedDTC= 0x08	bit 3: ConfirmedDTC
	kTestNotCompletedSinceLastClear= 0x10	bit 4: TestNotCompletedSinceLastClear
	kTestFailedSinceLastClear= 0x20	bit 5: TestFailedSinceLastClear
	kTestNotCompletedThisOperation Cycle= 0x40	bit 6: TestNotCompletedThisOperationCycle
	kWarningIndicatorRequested= 0x80	bit 7: WarningIndicatorRequested
Header file:	#include "ara/diag/dtc_information.h"	
Description:	UDS DTC status bits according to ISO 14229-1.	

⌋(*RS_Diag_04151*, *RS_Diag_04067*)

8.5.2.3 diag::DTCInformation::UdsDtcStatusByteType type

[SWS_DM_00659]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::DTCInformation::UdsDtcStatusByteType
Scope:	class ara::diag::DTCInformation
Syntax:	`struct UdsDtcStatusByteType {...};`
Header file:	#include "ara/diag/dtc_information.h"
Description:	Type for UDS DTC status byte.

⌋(*RS_Diag_04151*, *RS_Diag_04067*)

8.5.2.4 diag::DTCInformation::SnapshotDataIdentiferType type

[SWS_DM_00660]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::DTCInformation::SnapshotDataIdentiferType
Scope:	class ara::diag::DTCInformation
Syntax:	`struct SnapshotDataIdentiferType {...};`
Header file:	#include "ara/diag/dtc_information.h"
Description:	Type for SnapshotDataIdentiferType status.

⌋(*RS_Diag_04205*)

8.5.2.5 diag::DTCInformation::SnapshotDataRecordType type

[SWS_DM_00661]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::DTCInformation::SnapshotDataRecordType
Scope:	class ara::diag::DTCInformation
Syntax:	`struct SnapshotDataRecordType {...};`
Header file:	#include "ara/diag/dtc_information.h"
Description:	Type for SnapshotDataRecordType status.

⌋(*RS_Diag_04205*)

8.5.2.6 diag::DTCInformation::SnapshotRecordUpdatedType type

[SWS_DM_00662]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::DTCInformation::SnapshotRecordUpdatedType
Scope:	class ara::diag::DTCInformation
Syntax:	`struct SnapshotRecordUpdatedType {...};`
Header file:	#include "ara/diag/dtc_information.h"
Description:	Type for SnapshotRecordUpdatedType status.

⌋(*RS_Diag_04205*)

8.5.2.7 diag::DTCInformation::DTCInformation function

[SWS_DM_00664]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::DTCInformation(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::DTCInformation	
Syntax:	`explicit DTCInformation (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticDTCInformationInterface
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Constructor for a DTCInformation instance which allows for DTC related operation per DiagnosticMemoryDestination.	

⌋(*RS_AP_00137*, *RS_Diag_04150*, *RS_Diag_04164*, *RS_Diag_04105*)

8.5.2.8 diag::DTCInformation::~DTCInformation function

[SWS_DM_00665]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DTCInformation::~DTCInformation()
Scope:	class ara::diag::DTCInformation
Syntax:	`~DTCInformation () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/dtc_information.h"
Description:	Destructor of class DTCInformation.

⌋(*RS_AP_00134*)

8.5.2.9 diag::DTCInformation::GetCurrentStatus function

[SWS_DM_00666]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::GetCurrentStatus(std::uint32_t dtc)	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<UdsDtcStatusByteType> GetCurrentStatus (std::uint32_t dtc);`	
Parameters (in):	dtc	DTC indentifier for which the status should be retrieved.
Return value:	ara::core::Result< UdsDtcStatusByteType >	the current UDS DTC status byte of the given DTC identifier.
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Retrieves the current UDS DTC status byte of the given DTC identifier.	

⌋(*RS_AP_00139*)

8.5.2.10 diag::DTCInformation::SetDTCStatusChangedNotifier function

[SWS_DM_00667]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::SetDTCStatusChangedNotifier(std::Function< void(std::uint32_t dtc, ara::diag::UdsDtcStatusByteType udsStatusByteOld, ara::diag::UdsDtcStatusByteType udsStatusByteNew)> notifier)	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<void> SetDTCStatusChangedNotifier (std::Function< void(std::uint32_t dtc, ara::diag::UdsDtcStatusByteType udsStatusByteOld, ara::diag::UdsDtcStatusByteType udsStatusByteNew)> notifier);`	
Parameters (in):	notifier	The function to be called if a DTC status has changed.
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Register a notifier function which is called if a UDS DTC status is changed.	

⌋(*RS_Diag_04148*, *RS_AP_00139*)

8.5.2.11 diag::DTCInformation::SetSnapshotRecordUpdatedNotifier function

[SWS_DM_00668]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DTCInformation::SetSnapshotRecordUpdatedNotifier(std::Function< void(ara::diag::SnapshotRecordUpdatedType)> notifier)
Scope:	class ara::diag::DTCInformation
Syntax:	`ara::core::Result<void> SetSnapshotRecordUpdatedNotifier (std::Function< void(ara::diag::SnapshotRecordUpdatedType)> notifier);`

▽

△

Parameters (in):	notifier	The function to be called if the SnapshotRecord is changed.
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Register a notifier function which is called if the SnapshotRecord is changed.	

⌋(*RS_Diag_04205*, *RS_AP_00139*)

8.5.2.12 diag::DTCInformation::GetNumberOfStoredEntries function

[SWS_DM_00669]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::GetNumberOfStoredEntries()	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<std::uint32_t> GetNumberOfStoredEntries ();`	
Return value:	ara::core::Result< std::uint32_t >	Number of currently stored fault memory entries.
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Contains the number of currently stored fault memory entries.	

⌋(*RS_Diag_04109*, *RS_AP_00139*)

8.5.2.13 diag::DTCInformation::SetNumberOfStoredEntriesNotifier function

[SWS_DM_00670]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::SetNumberOfStoredEntriesNotifier(std::Function< void(std::uint32_t)> notifier)	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<void> SetNumberOfStoredEntriesNotifier (std::Function< void(std::uint32_t)> notifier);`	
Parameters (in):	notifier	The function to be called if the number of entries for this diagnostic memory instance has changed.
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Register a notifier function which is called if the number of currently stored fault memory entries changed.	

⌋(*RS_Diag_04109*, *RS_AP_00139*)

8.5.2.14 diag::DTCInformation::Clear function

[SWS_DM_00671]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::Clear(std::uint32_t DTCGroup)	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<void> Clear (std::uint32_t DTCGroup);`	
Parameters (in):	DTCGroup	DTC group to be cleared.
Return value:	ara::core::Result< void >	void or errors
Errors:	DiagErrorDomain::DiagErrc::kBusy	Busy processing.
	DiagErrorDomain::DiagErrc::kFailed	Clear failed.
	DiagErrorDomain::DiagErrc::kMemory Error	Memory error reported.
	DiagErrorDomain::DiagErrc::kWrong Dtc	Wrong DTC group passed.
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Method for Clearing a DTC or a group of DTCs.	

⌋(*RS_Diag_04194*, *RS_AP_00139*)

8.5.2.15 diag::DTCInformation::GetControlDTCStatus function

[SWS_DM_00672]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::GetControlDTCStatus()	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<ControlDtcStatusType> GetControlDTCStatus ();`	
Return value:	ara::core::Result< ControlDtcStatus Type >	The current status of ControlDtcStatus (related to UDS service 0x85) or an UDS NRC value (for an negative response message)
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Contains the current status of the ControlDTCStatus.	

⌋(*RS_Diag_04159*, *RS_AP_00139*)

8.5.2.16 diag::DTCInformation::SetControlDtcStatusNotifier function

[SWS_DM_00673]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::SetControlDtcStatusNotifier(std::Function< void(ControlDtcStatus Type)> notifier)	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<void> SetControlDtcStatusNotifier (std::Function< void(ControlDtcStatusType)> notifier);`	
Parameters (in):	notifier	The function to be called if the ControlDTCStatus (related to UDS service 0x85) for this diagnostic memory instance has changed.
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Registers a notifier function which is called if the control DTC setting is changed.	

⌋(*RS_Diag_04159*, *RS_AP_00139*)

8.5.2.17 diag::DTCInformation::EnableControlDtc function

[SWS_DM_00674]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DTCInformation::EnableControlDtc()	
Scope:	class ara::diag::DTCInformation	
Syntax:	`ara::core::Result<void> EnableControlDtc ();`	
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/dtc_information.h"	
Description:	Enforce restoring ControlDTCStatus setting to enabled in case the monitor has some conditions or states demands to do so.	

⌋(*RS_Diag_04159*, *RS_AP_00139*)

8.5.3 Conversation class

This interface is replacing the obsolete `DiagnosticConversation` service interface. The conversation object can only be retrieved by a given meta_info object.

[SWS_DM_00693]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::Conversation
Scope:	namespace ara::diag
Syntax:	`class Conversation {...};`
Header file:	#include "ara/diag/conversation.h"

▽

△

Description:	Conversation interface.

⌋()

8.5.3.1 diag::Conversation::ActivityStatusType type

[SWS_DM_00690]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::Conversation::ActivityStatusType	
Scope:	class ara::diag::Conversation	
Underlying type:	–	
Syntax:	`enum class ActivityStatusType {...};`	
Values:	kActive= 0x00	Currently active; i.e. request is currently processed or non-default session is active.
	kInactive= 0x01	Currently not active.
Header file:	#include "ara/diag/conversation.h"	
Description:	Type for current activity status.	

⌋()

8.5.3.2 diag::Conversation::ConversationIdentifierType type

[SWS_DM_00691]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::Conversation::ConversationIdentifierType
Scope:	class ara::diag::Conversation
Syntax:	`struct ConversationIdentifierType {...};`
Header file:	#include "ara/diag/conversation.h"
Description:	Properties allowing an identification of the conversation.

⌋()

8.5.3.3 diag::Conversation::GetConversation function

[SWS_DM_00692]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::GetConversation(ara::diag::MetaInfo meta_info)	
Scope:	class ara::diag::Conversation	
Syntax:	`static ara::core::Result<ara::diag::Conversation&> GetConversation (ara::diag::MetaInfo meta_info);`	
Parameters (in):	meta_info	contains additional meta information
Return value:	ara::core::Result< ara::diag::Conversation & >	Conversation object or error
Header file:	#include "ara/diag/conversation.h"	
Description:	Get one conversation based on given MetaInfo.	

⌋(*RS_AP_00139*, *RS_Diag_04170*)

8.5.3.4 diag::Conversation::GetAllConversations function

[SWS_DM_00782]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::GetAllConversations()	
Scope:	class ara::diag::Conversation	
Syntax:	`static ara::core::Vector<ara::diag::Conversation&> GetAllConversations ();`	
Return value:	ara::core::Vector< ara::diag::Conversation & >	a vector of all possibe Conversation objects
Header file:	#include "ara/diag/conversation.h"	
Description:	Get all possible conversations.	

⌋()

8.5.3.5 diag::Conversation::GetCurrentActiveConversations function

[SWS_DM_00783]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::GetCurrentActiveConversations()	
Scope:	class ara::diag::Conversation	
Syntax:	`static ara::core::Vector<ara::diag::Conversation&> GetCurrentActive Conversations ();`	
Return value:	ara::core::Vector< ara::diag::Conversation & >	a vector of all currently active (GetActivityStatus() == kActive) Conversation objects
Header file:	#include "ara/diag/conversation.h"	
Description:	Get all currently active conversations.	

⌋()

8.5.3.6 diag::Conversation::GetActivityStatus function

[SWS_DM_00694]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::GetActivityStatus()	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<ara::diag::ActivityStatusType> GetActivityStatus ();`	
Return value:	ara::core::Result< ara::diag::Activity StatusType >	the activity status of the conversation
Header file:	#include "ara/diag/conversation.h"	
Description:	Represents the status of an active conversation.	

⌋(*RS_AP_00139*)

8.5.3.7 diag::Conversation::SetActivityNotifier function

[SWS_DM_00695]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::SetActivityNotifier(std::function< void(ara::diag::ActivityStatusType)> notifier)	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<void> SetActivityNotifier (std::function< void(ara::diag::ActivityStatusType)> notifier);`	
Parameters (in):	notifier	notifier function to be called
Return value:	ara::core::Result< void >	void when the registering went fine or error
Header file:	#include "ara/diag/conversation.h"	
Description:	Register a notifier function which is called if the activity is changed.	

⌋(*RS_AP_00139*)

8.5.3.8 diag::Conversation::GetConversationIdentifier function

[SWS_DM_00700]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::GetConversationIdentifier()	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<ara::diag::ConversationIdentifierType> GetConversationIdentifier ();`	
Return value:	ara::core::Result< ara::diag::ConversationIdentifierType >	the conversation information

▽

△

Header file:	#include "ara/diag/conversation.h"
Description:	Getter for the current identification properties of the active conversation.

⌋(*RS_AP_00139*)

8.5.3.9 diag::Conversation::GetDiagnosticSession function

[SWS_DM_00696]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::GetDiagnosticSession()	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<ara::core::StringView> GetDiagnosticSession ();`	
Return value:	ara::core::Result< ara::core::String View >	the current session
Header file:	#include "ara/diag/conversation.h"	
Description:	Represents the current active diagnostic session of an active conversation.	

⌋(*RS_AP_00139*)

8.5.3.10 diag::Conversation::SetDiagnosticSessionNotifier function

[SWS_DM_00697]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::SetDiagnosticSessionNotifier(std::Function< void(ara::core::String View)> notifier)	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<void> SetDiagnosticSessionNotifier (std::Function< void(ara::core::StringView)> notifier);`	
Parameters (in):	notifier	notifier function to be called
Return value:	ara::core::Result< void >	void when the registering went fine or error
Header file:	#include "ara/diag/conversation.h"	
Description:	Register a notifier function which is called if the Session is changed.	

⌋(*RS_AP_00139*)

8.5.3.11 diag::Conversation::GetDiagnosticSecurityLevel function

[SWS_DM_00698]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::GetDiagnosticSecurityLevel()	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<ara::core::StringView> GetDiagnosticSecurityLevel ();`	
Return value:	ara::core::Result< ara::core::String View >	the current SecurityLevel
Header file:	#include "ara/diag/conversation.h"	
Description:	Represents the current active diagnostic SecurityLevel of an active conversation.	

⌋(*RS_AP_00139*)

8.5.3.12 diag::Conversation::SetSecurityLevelNotifier function

[SWS_DM_00699]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::SetSecurityLevelNotifier(std::Function< void(ara::core::StringView)> notifier)	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<void> SetSecurityLevelNotifier (std::Function< void(ara::core::StringView)> notifier);`	
Parameters (in):	notifier	notifier function to be called
Return value:	ara::core::Result< void >	void when the registering went fine or error
Header file:	#include "ara/diag/conversation.h"	
Description:	Register a notifier function which is called if the SecurityLevel is changed.	

⌋(*RS_AP_00139*)

8.5.3.13 diag::Conversation::ResetToDefaultSession function

[SWS_DM_00701]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::ResetToDefaultSession()	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<void> ResetToDefaultSession ();`	
Return value:	ara::core::Result< void >	void on success or error
Header file:	#include "ara/diag/conversation.h"	
Description:	Method to reset the current session to default session.	

⌋()

8.5.3.14 diag::Conversation::Cancel function

[SWS_DM_00702]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Conversation::Cancel()	
Scope:	class ara::diag::Conversation	
Syntax:	`ara::core::Result<void> Cancel ();`	
Return value:	ara::core::Result< void >	void on success or error
Header file:	#include "ara/diag/conversation.h"	
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.	

⌋()

8.5.4 Condition class

This interface is replacing the obsolete `EnableCondition` and `ClearCondition` service interfaces.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticConditionInterface`.

[SWS_DM_00711]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::Condition
Scope:	namespace ara::diag
Syntax:	`class Condition {...};`
Header file:	#include "ara/diag/condition.h"
Description:	DiagnosticConditionInterface.

⌋()

8.5.4.1 diag::Condition::ConditionType type

[SWS_DM_00710]{DRAFT} ⌈

Kind:	enumeration
Symbol:	ara::diag::Condition::ConditionType
Scope:	class ara::diag::Condition
Underlying type:	–
Syntax:	`enum class ConditionType {...};`

▽

△

Values:	kConditionFalse= 0x00	condition is set to false
	kConditionTrue= 0x01	condition is set to true
Header file:	#include "ara/diag/condition.h"	
Description:	Type for Condition status.	

⌋()

8.5.4.2 diag::Condition::Condition function

[SWS_DM_00712]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Condition::Condition(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::Condition	
Syntax:	`explicit Condition (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticConditionInterface
Header file:	#include "ara/diag/condition.h"	
Description:	Constructor of Condition Class.	

⌋(*RS_AP_00137*)

8.5.4.3 diag::Condition::~Condition function

[SWS_DM_00713]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::Condition::~Condition()
Scope:	class ara::diag::Condition
Syntax:	`~Condition () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/condition.h"
Description:	Destructor of class Condition.

⌋(*RS_AP_00134*)

8.5.4.4 diag::Condition::GetCurrentStatus function

[SWS_DM_00714]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Condition::GetCondition()	
Scope:	class ara::diag::Condition	
Syntax:	`ara::core::Result<ConditionType> GetCondition ();`	
Return value:	ara::core::Result< ConditionType >	the current condition
Header file:	#include "ara/diag/condition.h"	
Description:	Get current condition.	

⌋(*RS_AP_00139*)

8.5.4.5 diag::Condition::SetCondition function

[SWS_DM_00715]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Condition::SetCondition(ara::diag::ConditionType condition)	
Scope:	class ara::diag::Condition	
Syntax:	`ara::core::Result<void> SetCondition (ara::diag::ConditionType condition);`	
Parameters (in):	condition	current condition
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/condition.h"	
Description:	Set condition.	

⌋(*RS_AP_00139*)

8.5.5 OperationCycle class

This interface is replacing the obsolete `OperationCycle` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticOperationCycleInterface`.

[SWS_DM_00751]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::OperationCycle
Scope:	namespace ara::diag
Syntax:	`class OperationCycle {...};`
Header file:	#include "ara/diag/operation_cycle.h"
Description:	DiagnosticOperationCycleInterface provides functionality for handling of operation cycles.

⌋(*RS_Diag_04178*)

8.5.5.1 diag::OperationCycle::OperationCycleType type

[SWS_DM_00750]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::OperationCycle::OperationCycleType	
Scope:	class ara::diag::OperationCycle	
Underlying type:	–	
Syntax:	`enum class OperationCycleType {...};`	
Values:	kOperationCycleStart= 0x00	Start/restart the operation cycle.
	kOperationCycleEnd= 0x01	End the operation cycle.
Header file:	#include "ara/diag/operation_cycle.h"	
Description:	Represents the state information of operation cycles.	

⌋(*RS_Diag_04178*)

8.5.5.2 diag::OperationCycle::OperationCycle function

[SWS_DM_00752]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::OperationCycle::OperationCycle(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::OperationCycle	
Syntax:	`explicit OperationCycle (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticOperationCycleInterface
Header file:	#include "ara/diag/operation_cycle.h"	
Description:	Constructor for DiagnosticOperationCycleInterface.	

⌋(*RS_AP_00137*, *RS_Diag_04178*)

8.5.5.3 diag::OperationCycle::~OperationCycle function

[SWS_DM_00753]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::OperationCycle::~OperationCycle()
Scope:	class ara::diag::OperationCycle
Syntax:	`~OperationCycle () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/operation_cycle.h"

▽

△

Description:	Destructor of DiagnosticOperationCycleInterface.

⌋(*RS_AP_00134*, *RS_Diag_04178*)

8.5.5.4 diag::OperationCycle::GetOperationCycle function

[SWS_DM_00754]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::OperationCycle::GetOperationCycle()	
Scope:	class ara::diag::OperationCycle	
Syntax:	`ara::core::Result<OperationCycleType> GetOperationCycle ();`	
Return value:	ara::core::Result< OperationCycleType >	the current OperationCycle
Header file:	#include "ara/diag/operation_cycle.h"	
Description:	Get current OperationCycle.	

⌋(*RS_AP_00139*, *RS_Diag_04178*)

8.5.5.5 diag::OperationCycle::SetNotifier function

[SWS_DM_00755]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::OperationCycle::SetNotifier(std::Function< void(OperationCycleType)> notifier)	
Scope:	class ara::diag::OperationCycle	
Syntax:	`ara::core::Result<void> SetNotifier (std::Function< void(OperationCycleType)> notifier);`	
DIRECTION NOT DEFINED	notifier	–
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/operation_cycle.h"	
Description:	Registering a notifier function which is called if the operation cycle is changed.	

⌋(*RS_AP_00139*, *RS_Diag_04178*, *RS_Diag_04186*)

8.5.5.6 diag::OperationCycle::SetOperationCycle function

[SWS_DM_00756]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::OperationCycle::SetOperationCycle(ara::diag::OperationCycleType operation_cycle)	
Scope:	class ara::diag::OperationCycle	
Syntax:	`ara::core::Result<void> SetOperationCycle (ara::diag::OperationCycle Type operation_cycle);`	
Parameters (in):	operation_cycle	current OperationCycle
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/operation_cycle.h"	
Description:	Set OperationCycle.	

⌋(*RS_AP_00139*, *RS_Diag_04178*, *RS_Diag_04182*)

8.5.6 Indicator class

This interface is replacing the obsolete `Indicator` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticIndicatorInterface`.

[SWS_DM_00741]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::Indicator
Scope:	namespace ara::diag
Syntax:	`class Indicator {...};`
Header file:	#include "ara/diag/indicator.h"
Description:	DiagnosticIndicatorInterface provides functionality for handling indicators.

⌋(*RS_Diag_04204*)

8.5.6.1 diag::Indicator::IndicatorType type

[SWS_DM_00740]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::Indicator::IndicatorType	
Scope:	class ara::diag::Indicator	
Underlying type:	–	
Syntax:	`enum class IndicatorType {...};`	
Values:	kOff= 0x00	Indicator off mode {default}.
	kContinouse= 0x01	Indicator continuously on mode.
	kBlinking= 0x02	Indicator blinking mode.

▽

△

	kBlinkingAndContinouse= 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.
Header file:	#include "ara/diag/indicator.h"	
Description:	Represents the state of an indicator.	

⌋(*RS_Diag_04204*)

8.5.6.2 diag::Indicator::Indicator function

[SWS_DM_00742]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Indicator::Indicator(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::Indicator	
Syntax:	`explicit Indicator (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticIndicatorInterface
Header file:	#include "ara/diag/indicator.h"	
Description:	Constructor for DiagnosticIndicatorInterface.	

⌋(*RS_AP_00137*, *RS_Diag_04204*)

8.5.6.3 diag::Indicator::~Indicator function

[SWS_DM_00743]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::Indicator::~Indicator()
Scope:	class ara::diag::Indicator
Syntax:	`~Indicator () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/indicator.h"
Description:	Destructor of DiagnosticIndicatorInterface.

⌋(*RS_AP_00134*, *RS_Diag_04204*)

8.5.6.4 diag::Indicator::GetIndicator function

[SWS_DM_00744]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Indicator::GetIndicator()	
Scope:	class ara::diag::Indicator	
Syntax:	`ara::core::Result<IndicatorType> GetIndicator ();`	
Return value:	ara::core::Result< IndicatorType >	the current Indicator
Header file:	#include "ara/diag/indicator.h"	
Description:	Get current Indicator.	

⌋(*RS_AP_00139*, *RS_Diag_04204*)

8.5.6.5 diag::Indicator::SetNotifier function

[SWS_DM_00745]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::Indicator::SetNotifier(std::Function< void(IndicatorType)> notifier)	
Scope:	class ara::diag::Indicator	
Syntax:	`ara::core::Result<void> SetNotifier (std::Function< void(Indicator Type)> notifier);`	
Parameters (in):	notifier	notifier function
Return value:	ara::core::Result< void >	–
Header file:	#include "ara/diag/indicator.h"	
Description:	Register a notifier function which is called if the indicator is updated.	

⌋(*RS_AP_00139*, *RS_Diag_04204*)

8.5.7 ServiceValidation class

This interface is replacing the obsolete `ServiceManufacturerValidation` and `ServiceSupplierValidation` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticService-ValidationInterface`.

[SWS_DM_00771]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::ServiceValidation
Scope:	namespace ara::diag

▽

△

Syntax:	`class ServiceValidation {...};`
Header file:	#include "ara/diag/service_validation.h"
Description:	DiagnosticServiceValidationInterface.

⌋(*RS_Diag_04199*)

8.5.7.1 diag::ServiceValidation::ConfirmationStatusType

[SWS_DM_00770]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::ServiceValidation::ConfirmationStatusType	
Scope:	class ara::diag::ServiceValidation	
Underlying type:	–	
Syntax:	`enum class ConfirmationStatusType {...};`	
Values:	kResPosOk= 0x00	Positive response has been sent out successfully.
	kResPosNotOk= 0x01	Positive response has not been sent out successfully.
	kResNegOk= 0x02	Negative response has been sent out successfull.
	kResNegNotOk= 0x03	Negative response has not been sent out successfully.
	kResPosSuppressed= 0x04	Positive answer suppressed.
	kResNegSuppressed= 0x05	Negative answer suppressed.
	kCanceled= 0x06	Processing is canceled.
	kNoProcessingNoResponse= 0x07	Processing rejected in Validation.
Header file:	#include "ara/diag/service_validation.h"	
Description:	Represents the status of the service processing.	

⌋(*RS_Diag_04199*)

8.5.7.2 diag::ServiceValidation::ServiceValidation function

[SWS_DM_00772]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::ServiceValidation::ServiceValidation(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::ServiceValidation	
Syntax:	`explicit ServiceValidation (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticServiceValidationInterface

▽

△

Header file:	#include "ara/diag/service_validation.h"
Description:	Constructor of ServiceValidation.

⌋(*RS_AP_00137*, *RS_Diag_04199*)

8.5.7.3 diag::ServiceValidation::~ServiceValidation function

[SWS_DM_00773]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::ServiceValidation::~ServiceValidation()
Scope:	class ara::diag::ServiceValidation
Syntax:	`virtual ~ServiceValidation () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/service_validation.h"
Description:	Destructor of ServiceValidation.

⌋(*RS_AP_00134*, *RS_Diag_04199*)

8.5.7.4 diag::ServiceValidation::Validate function

[SWS_DM_00774]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::ServiceValidation::Validate(ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info)	
Scope:	class ara::diag::ServiceValidation	
Syntax:	`virtual ara::core::Future<void> Validate (ara::core::Span< std::uint8_t > request_data, ara::diag::MetaInfo meta_info);`	
Parameters (in):	request_data	Diagnostic request data (including SID).
	meta_info	MetaInfo of the request.
Return value:	ara::core::Future< void >	Returns nothing or an error
Errors:	tbd	This error set includes all NegativeResponseCodes defined in UDS.
Header file:	#include "ara/diag/service_validation.h"	
Description:	Called for any request messsage.	

⌋(*RS_AP_00138*, *RS_Diag_04170*, *RS_Diag_04199*)

8.5.7.5 diag::ServiceValidation::Confirmation function

[SWS_DM_00775]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::ServiceValidation::Confirmation(ara::diag::ConfirmationStatusType status, ara::diag::MetaInfo meta_info)	
Scope:	class ara::diag::ServiceValidation	
Syntax:	`virtual ara::core::Future<void> Confirmation (ara::diag::Confirmation StatusType status, ara::diag::MetaInfo meta_info);`	
Parameters (in):	status	status/outcome of the service processing.
	meta_info	MetaInfo of the request.
Return value:	ara::core::Future< void >	Returns nothing or an error
Header file:	#include "ara/diag/service_validation.h"	
Description:	This method is called, when a diagnostic request has been finished, to notify about the outcome.	

⌋(*RS_AP_00138*, *RS_Diag_04170*, *RS_Diag_04199*)

8.5.7.6 diag::ServiceValidation::Offer function

[SWS_DM_00776]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::ServiceValidation::Offer()	
Scope:	class ara::diag::ServiceValidation	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	Returns nothing or an error
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/service_validation.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*, *RS_Diag_04199*)

8.5.7.7 diag::ServiceValidation::StopOffer function

[SWS_DM_00777]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::ServiceValidation::StopOffer()
Scope:	class ara::diag::ServiceValidation
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/service_validation.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋(*RS_Diag_04199*)

8.5.8 SecurityAccess class

This interface is replacing the obsolete `SecurityAccess` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticSecurityLevelInterface`.

[SWS_DM_00761]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::SecurityAccess
Scope:	namespace ara::diag
Syntax:	`class SecurityAccess {...};`
Header file:	#include "ara/diag/security_access.h"
Description:	DiagnosticSecurityAccessInterface.

⌋*(RS_Diag_04005)*

8.5.8.1 diag::SecurityAccess::KeyCompareResultType type

[SWS_DM_00760]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::SecurityAccess::KeyCompareResultType	
Scope:	class ara::diag::SecurityAccess	
Underlying type:	–	
Syntax:	`enum class KeyCompareResultType {...};`	
Values:	kKeyValid= 0x00	Key is valid.
	kKeyInvalid= 0x01	Key is invalid.
Header file:	#include "ara/diag/security_access.h"	
Description:	Represents the status of the key compare.	

⌋*(RS_Diag_04005)*

8.5.8.2 diag::SecurityAccess::SecurityAccess function

[SWS_DM_00762]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::SecurityAccess::SecurityAccess(const ara::core::InstanceSpecifier &specifier)
Scope:	class ara::diag::SecurityAccess
Syntax:	`explicit SecurityAccess (const ara::core::InstanceSpecifier &specifier);`

▽

△

Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticSecurityAccessInterface
Header file:	#include "ara/diag/security_access.h"	
Description:	Constructor of SecurityAccess.	

⌋(*RS_AP_00137*, *RS_Diag_04005*)

8.5.8.3 diag::SecurityAccess::~SecurityAccess function

[SWS_DM_00763]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::SecurityAccess::~SecurityAccess()
Scope:	class ara::diag::SecurityAccess
Syntax:	`virtual ~SecurityAccess () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/security_access.h"
Description:	Destructor of SecurityAccess.

⌋(*RS_AP_00134*, *RS_Diag_04005*)

8.5.8.4 diag::SecurityAccess::GetSeed function

[SWS_DM_00764]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::SecurityAccess::GetSeed(ara::core::Span< std::uint8_t > security_access_data_record, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::SecurityAccess	
Syntax:	`virtual ara::core::Future<ara::core::Span<std::uint8_t> > GetSeed (ara::core::Span< std::uint8_t > security_access_data_record, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	security_access_data_record	Security Access payload
	meta_info	MetaInfo of the request.
	cancellation_handler	Set if the current conversation is canceled.
Return value:	ara::core::Future< ara::core::Span< std::uint8_t > >	provided seed
Errors:	tbd	This error set includes all NegativeResponseCodes defined in UDS.
Header file:	#include "ara/diag/security_access.h"	
Description:	Called for any request messsage.	

⌋(*RS_AP_00138*, *RS_Diag_04005*, *RS_Diag_04170*)

8.5.8.5 diag::SecurityAccess::CompareKey function

[SWS_DM_00765]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::SecurityAccess::CompareKey(ara::core::Span< std::uint8_t > key, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::SecurityAccess	
Syntax:	`virtual ara::core::Future<ara::diag::KeyCompareResultType> CompareKey (ara::core::Span< std::uint8_t > key, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	key	The key to be validated
	meta_info	MetaInfo of the request.
	cancellation_handler	Set if the current conversation is canceled.
Return value:	ara::core::Future< ara::diag::KeyCompareResultType >	Result of the key validation.
Errors:	tbd	This error set includes all NegativeResponseCodes defined in UDS.
Header file:	#include "ara/diag/security_access.h"	
Description:	This method is called, when a diagnostic request has been finished, to notify about the outcome.	

⌋(*RS_AP_00138*, *RS_Diag_04005*, *RS_Diag_04170*)

8.5.8.6 diag::SecurityAccess::Offer function

[SWS_DM_00766]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::SecurityAccess::Offer()	
Scope:	class ara::diag::SecurityAccess	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/security_access.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*, *RS_Diag_04005*)

8.5.8.7 diag::SecurityAccess::StopOffer function

[SWS_DM_00767]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::SecurityAccess::StopOffer()
Scope:	class ara::diag::SecurityAccess
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/security_access.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋(*RS_Diag_04005*)

8.5.9 CommunicationControl class

This interface is replacing the obsolete `CommunicationControl` service interface. The InstanceSpecifier is only compatible with PortInterface of *DiagnosticCommunicationControlInterface*.

[SWS_DM_00804]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::CommunicationControl
Scope:	namespace ara::diag
Syntax:	`class CommunicationControl {...};`
Header file:	#include "ara/diag/communication_control.h"
Description:	CommunicationControl interface.

⌋()

8.5.9.1 diag::CommunicationControl::ComCtrlRequestParamsType type

[SWS_DM_00805]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::CommunicationControl::ComCtrlRequestParamsType
Scope:	class ara::diag::CommunicationControl
Syntax:	`struct ComCtrlRequestParamsType {...};`
Header file:	#include "ara/diag/communication_control.h"
Description:	ComCtrlRequestParamsType is a structure, which holds all parameters of an UDS 0x28 communicationControl request.

⌋() CommunicationControl

8.5.9.2 diag::CommunicationControl::CommunicationControl function

[SWS_DM_00806]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::CommunicationControl::CommunicationControl(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::CommunicationControl	
Syntax:	`explicit CommunicationControl (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticCommunicationControlInterface
Header file:	#include "ara/diag/communication_control.h"	
Description:	Class for an CommunicationControl.	

⌋(*RS_AP_00137*)

8.5.9.3 diag::CommunicationControl::~CommunicationControl function

[SWS_DM_00807]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::CommunicationControl::~CommunicationControl()
Scope:	class ara::diag::CommunicationControl
Syntax:	`virtual ~CommunicationControl () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/communication_control.h"
Description:	Destructor of class CommunicationControl.

⌋(*RS_AP_00134*)

8.5.9.4 diag::CommunicationControl::CommCtrlRequest function

[SWS_DM_00808]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::CommunicationControl::CommCtrlRequest(ara::diag::ComCtrlRequestParamsType controlType, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)
Scope:	class ara::diag::CommunicationControl
Syntax:	`virtual ara::core::Future<void> CommCtrlRequest (ara::diag::ComCtrlRequestParamsType controlType, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`

▽

△

Parameters (in):	controlType	All UDS request parameters packed into a structure since it holds optional elements
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< void >	–
Errors:	tbd	Any applicable NegativeResponseValue
Header file:	#include "ara/diag/communication_control.h"	
Description:	Called for CommunicationControl (x028) with any subfunction as subfunction value is part of argument list. Typically provider of this interface is considered as part of the state management.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.5.9.5 diag::CommunicationControl::Offer function

[SWS_DM_00809]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::CommunicationControl::Offer()	
Scope:	class ara::diag::CommunicationControl	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/communication_control.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*)

8.5.9.6 diag::CommunicationControl::StopOffer function

[SWS_DM_00810]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::CommunicationControl::StopOffer()
Scope:	class ara::diag::CommunicationControl
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/communication_control.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.5.10 DownloadService class

This interface is newly introduced.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticDownloadInterface`.

[SWS_DM_00784]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::DownloadService
Scope:	namespace ara::diag
Syntax:	`class DownloadService {...};`
Header file:	#include "ara/diag/download.h"
Description:	Download service interface.

⌋()

8.5.10.1 diag::DownloadService::OperationOutput type

[SWS_DM_00785]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::DownloadService::OperationOutput
Scope:	class ara::diag::DownloadService
Syntax:	`struct OperationOutput {...};`
Header file:	#include "ara/diag/download.h"
Description:	Response data of positive respone message.

⌋()

8.5.10.2 diag::DownloadService::DownloadServicefunction

[SWS_DM_00787]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DownloadService::DownloadService(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::DownloadService	
Syntax:	`explicit DownloadService (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DownloadServiceInterface
Header file:	#include "ara/diag/download.h"	

▽

△

Description:	Class for an DownloadService.

⌋(*RS_AP_00137*)

8.5.10.3 diag::DownloadService::~DownloadServicefunction

[SWS_DM_00788]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DownloadService::~DownloadService()
Scope:	class ara::diag::DownloadService
Syntax:	`virtual ~DownloadService () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/download.h"
Description:	Destructor of class DownloadService.

⌋(*RS_AP_00134*)

8.5.10.4 diag::DownloadService::RequestDownload function

[SWS_DM_00789]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DownloadService::RequestDownload(std::uint8_t dataFormatIdentifier, std::uint8_t addressAndLengthFormatIdentifier, ara::core::Span< std::uint8_t > memoryAddressAndSize, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::DownloadService	
Syntax:	`virtual ara::core::Future<void> RequestDownload (std::uint8_t data FormatIdentifier, std::uint8_t addressAndLengthFormatIdentifier, ara::core::Span< std::uint8_t > memoryAddressAndSize, ara::diag::Meta Info meta_info, ara::diag::CancellationHandler cancellation_ handler)=0;`	
Parameters (in):	dataFormatIdentifier	UDS dataFormat Identifier
	addressAndLengthFormatIdentifier	UDS addressAndLengthFormatIdentifier
	memoryAddressAndSize	memoryAddress and memorySize part of the request
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< void >	a Future, which either gets readied to void (for a positive response message) or readied with Error Code from DiagUdsNrcErrc (for an negative response message)

▽

△

Errors:	Any	applicable NegativeResponseValue according to DiagUdsNrcErrc
Header file:	#include "ara/diag/download.h"	
Description:	Called for RequestDownload.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.5.10.5 diag::DownloadService::DownloadData function

[SWS_DM_00790]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DownloadService::DownloadData(ara::core::Span< std::uint8_t > transferRequest ParameterRecord, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_ handler)	
Scope:	class ara::diag::DownloadService	
Syntax:	`virtual ara::core::Future<OperationOutput> DownloadData (ara::core::Span< std::uint8_t > transferRequestParameterRecord, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	transferRequestParameterRecord	data to be transferred (copied/downloaded to the ECU/server).
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Future, which either gets readied to Operation Output (transferResponseParameterRecord for a positive response message) or readied with Error Code from DiagUdsNrcErrc (for an negative response message). Data in Operation Output.response_data will be placed after block SequenceCounter as transferResponseParameter Record in the positive response.
Errors:	Any	applicable NegativeResponseValue according to DiagUdsNrcErrc
Header file:	#include "ara/diag/download.h"	
Description:	Called for TransferData following a previous RequestDownload.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.5.10.6 diag::DownloadService::RequestDownloadExit function

[SWS_DM_00791]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DownloadService::RequestDownloadExit(ara::core::Span< std::uint8_t > transfer RequestParameterRecord, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::DownloadService	
Syntax:	`virtual ara::core::Future<OperationOutput> RequestDownloadExit (ara::core::Span< std::uint8_t > transferRequestParameterRecord, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	transferRequestParameterRecord	This parameter record contains parameter(s), which are required by the server to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific.
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Future, which either gets readied to Operation Output (transferResponseParameterRecord for a positive response message) or readied with Error Code from DiagUdsNrcErrc (for an negative response message) Data in Operation Output.response_data will be placed after SID as transferResponseParameterRecord in the positive response.
Errors:	Any	applicable NegativeResponseValue according to DiagUdsNrcErrc
Header file:	#include "ara/diag/download.h"	
Description:	Called for RequestTransferExit.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.5.10.7 diag::DownloadService::Offer function

[SWS_DM_00792]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DownloadService::Offer()	
Scope:	class ara::diag::DownloadService	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/download.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*)

8.5.10.8 diag::DownloadService::StopOffer function

[SWS_DM_00793]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DownloadService::StopOffer()
Scope:	class ara::diag::DownloadService
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/download.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.5.11 UploadService class

This interface is newly inroduced.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticUpload-Interface`.

[SWS_DM_00794]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::UploadService
Scope:	namespace ara::diag
Syntax:	`class UploadService {...};`
Header file:	#include "ara/diag/upload.h"
Description:	Upload service interface.

⌋()

8.5.11.1 diag::UploadService::OperationOutput type

[SWS_DM_00795]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::UploadService::OperationOutput
Scope:	class ara::diag::UploadService
Syntax:	`struct OperationOutput {...};`
Header file:	#include "ara/diag/upload.h"
Description:	Response data of positive respone message.

⌋()

8.5.11.2 diag::UploadService::UploadServicefunction

[SWS_DM_00797]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::UploadService::UploadService(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::UploadService	
Syntax:	`explicit UploadService (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DownloadServiceInterface
Header file:	#include "ara/diag/upload.h"	
Description:	Class for an UploadService.	

⌋(*RS_AP_00137*)

8.5.11.3 diag::UploadService::~UploadServicefunction

[SWS_DM_00798]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::UploadService::~UploadService()
Scope:	class ara::diag::UploadService
Syntax:	`virtual ~UploadService () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/upload.h"
Description:	Destructor of class UploadService.

⌋(*RS_AP_00134*)

8.5.11.4 diag::UploadService::RequestUpload function

[SWS_DM_00799]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::UploadService::RequestUpload(std::uint8_t dataFormatIdentifier, std::uint8_t addressAndLengthFormatIdentifier, ara::core::Span< std::uint8_t > memoryAddressAndSize, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::UploadService	
Syntax:	`virtual ara::core::Future<void> RequestUpload (std::uint8_t dataFormatIdentifier, std::uint8_t addressAndLengthFormatIdentifier, ara::core::Span< std::uint8_t > memoryAddressAndSize, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	dataFormatIdentifier	UDS dataFormat Identifier
	addressAndLengthFormatIdentifier	UDS addressAndLengthFormatIdentifier
	memoryAddressAndSize	memoryAddress and memorySize part of the request

▽

△

	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< void >	a Result with either void (for a positive response message) or an UDS NRC value (for an negative response message)
Errors:	Any	applicable NegativeResponseValue according to DiagUdsNrcErrc
Header file:	#include "ara/diag/upload.h"	
Description:	Called for RequestDownload.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.5.11.5 diag::UploadService::UploadData function

[SWS_DM_00800]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::UploadService::UploadData(std_size_t numBytesToReturn, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)	
Scope:	class ara::diag::UploadService	
Syntax:	`virtual ara::core::Future<OperationOutput> UploadData (std_size_t numBytesToReturn, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	numBytesToReturn	number of bytes DM accepts (due to its internal buffer) for this chunk.
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Future, which either gets readied to Operation Output (transferResponseParameterRecord for a positive response message) or readied with Error Code from DiagUdsNrcErrc (for an negative response message). Data in Operation Output.response_data will be placed after block SequenceCounter as transferResponseParameter Record in the positive response.
Errors:	Any	applicable NegativeResponseValue according to DiagUdsNrcErrc
Header file:	#include "ara/diag/upload.h"	
Description:	Called for TransferData following a previous RequestUpload.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.5.11.6 diag::UploadService::RequestUploadExit function

[SWS_DM_00801]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::UploadService::RequestUploadExit(ara::core::Span< std::uint8_t > transferRequest ParameterRecord, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_ handler)	
Scope:	class ara::diag::UploadService	
Syntax:	`virtual ara::core::Future<OperationOutput> RequestUploadExit (ara::core::Span< std::uint8_t > transferRequestParameterRecord, ara::diag::MetaInfo meta_info, ara::diag::CancellationHandler cancellation_handler)=0;`	
Parameters (in):	transferRequestParameterRecord	This parameter record contains parameter(s), which are required by the server to support the transfer of data. Format and length of this parameter(s) are vehicle manufacturer specific.
	meta_info	contains additional meta information
	cancellation_handler	informs if the current conversation is canceled
Return value:	ara::core::Future< OperationOutput >	a Future, which either gets readied to Operation Output (transferResponseParameterRecord for a positive response message) or readied with Error Code from DiagUdsNrcErrc (for an negative response message) Data in Operation Output.response_data will be placed after SID as transferResponseParameterRecord in the positive response.
Errors:	Any	applicable NegativeResponseValue according to DiagUdsNrcErrc
Header file:	#include "ara/diag/upload.h"	
Description:	Called for RequestTransferExit.	

⌋(*RS_AP_00138*, *RS_Diag_04170*)

8.5.11.7 diag::UploadService::Offer function

[SWS_DM_00802]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::UploadService::Offer()	
Scope:	class ara::diag::UploadService	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/upload.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*)

8.5.11.8 diag::UploadService::StopOffer function

[SWS_DM_00803]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::UploadService::StopOffer()
Scope:	class ara::diag::UploadService
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/upload.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋()

8.5.12 DoIPGroupIdentification class

This interface is replacing the obsolete `DoIPGroupIdentification` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticDoIP-GroupIdentificationInterface`.

[SWS_DM_00720]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::DoIPGroupIdentification
Scope:	namespace ara::diag
Syntax:	`class DoIPGroupIdentification {...};`
Header file:	#include "ara/diag/doip_group_identification.h"
Description:	DoIPGroupIdentificationInterface.

⌋(*SRS_Eth_00026*)

8.5.12.1 diag::DoIPGroupIdentification::DoIPGroupIdentificationType type

[SWS_DM_00721]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::diag::DoIPGroupIdentification::GidStatus
Scope:	class ara::diag::DoIPGroupIdentification
Syntax:	`struct GidStatus {...};`
Header file:	#include "ara/diag/doip_group_identification.h"
Description:	Response data of positive respone message.

⌋(*SRS_Eth_00026*)

8.5.12.2 diag::DoIPGroupIdentification::DoIPGroupIdentification function

[SWS_DM_00722]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPGroupIdentification::DoIPGroupIdentification(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::DoIPGroupIdentification	
Syntax:	`explicit DoIPGroupIdentification (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticDoIPGroupIdentificationInterface
Header file:	#include "ara/diag/doip_group_identification.h"	
Description:	Constructor of DoIPGroupIdentification.	

⌋(*RS_AP_00137*, *SRS_Eth_00026*)

8.5.12.3 diag::DoIPGroupIdentification::~DoIPGroupIdentification function

[SWS_DM_00723]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DoIPGroupIdentification::~DoIPGroupIdentification()
Scope:	class ara::diag::DoIPGroupIdentification
Syntax:	`virtual ~DoIPGroupIdentification () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/doip_group_identification.h"
Description:	Destructor of DoIPGroupIdentification.

⌋(*RS_AP_00134*, *SRS_Eth_00026*)

8.5.12.4 diag::DoIPGroupIdentification::GetGidStatus function

[SWS_DM_00724]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPGroupIdentification::GetGidStatus()	
Scope:	class ara::diag::DoIPGroupIdentification	
Syntax:	`virtual ara::core::Future<ara::diag::GidStatus> GetGidStatus ()=0;`	
DIRECTION NOT DEFINED	void	–
Return value:	ara::core::Future< ara::diag::GidStatus >	group identification and state

▽

△

Header file:	#include "ara/diag/doip_group_identification.h"
Description:	Called to get the current GID state for the DoIP protocol.

⌋(*RS_AP_00138*, *SRS_Eth_00026*)

8.5.12.5 diag::DoIPGroupIdentification::Offer function

[SWS_DM_00725]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPGroupIdentification::Offer()	
Scope:	class ara::diag::DoIPGroupIdentification	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/doip_group_identification.h"	
Description:	This Offer will enable the DM to forward request messages to this handler.	

⌋(*RS_AP_00139*, *SRS_Eth_00026*)

8.5.12.6 diag::DoIPGroupIdentification::StopOffer function

[SWS_DM_00726]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DoIPGroupIdentification::StopOffer()
Scope:	class ara::diag::DoIPGroupIdentification
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/doip_group_identification.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋(*SRS_Eth_00026*)

8.5.13 DoIPPowerMode class

This interface is replacing the obsolete `DoIPPowerModeInformation` service interface.
The InstanceSpecifier is only compatible with PortInterface of `DiagnosticDoIPPowerModeInterface`.

[SWS_DM_00731]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::DoIPPowerMode
Scope:	namespace ara::diag
Syntax:	`class DoIPPowerMode {...};`
Header file:	#include "ara/diag/doip_power_mode.h"
Description:	DiagnosticDoIPPowerModeInterface.

⌋(*SRS_Eth_00080*)

8.5.13.1 diag::DoIPPowerMode::PowerModeType type

[SWS_DM_00730]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::diag::DoIPPowerMode::PowerModeType	
Scope:	class ara::diag::DoIPPowerMode	
Underlying type:	–	
Syntax:	`enum class PowerModeType {...};`	
Values:	kNotReady= 0x00	not all ECUs accessible via DoIP can communicate
	kReady= 0x01	all ECUs accessible via DoIP can communicate
	kNotSupported= 0x02	the Diagnostic Information Power Mode Information Request message is not supported
Header file:	#include "ara/diag/doip_power_mode.h"	
Description:	PowerMode as defined in ISO13400-2.	

⌋(*SRS_Eth_00080*)

8.5.13.2 diag::DoIPPowerMode::DoIPPowerMode function

[SWS_DM_00732]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPPowerMode::DoIPPowerMode(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::DoIPPowerMode	
Syntax:	`explicit DoIPPowerMode (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticDoIPPowerModeInterface
Header file:	#include "ara/diag/doip_power_mode.h"	
Description:	Constructor of DoIPPowerMode.	

⌋(*RS_AP_00137*, *SRS_Eth_00080*)

8.5.13.3 diag::DoIPPowerMode::~DoIPPowerMode function

[SWS_DM_00733]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DoIPPowerMode::~DoIPPowerMode()
Scope:	class ara::diag::DoIPPowerMode
Syntax:	`virtual ~DoIPPowerMode () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/doip_power_mode.h"
Description:	Destructor of DoIPPowerMode.

⌋(*RS_AP_00134*, *SRS_Eth_00080*)

8.5.13.4 diag::DoIPPowerMode::GetDoIPPowerMode function

[SWS_DM_00734]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPPowerMode::GetDoIPPowerMode()	
Scope:	class ara::diag::DoIPPowerMode	
Syntax:	`virtual ara::core::Future<ara::diag::PowerModeType> GetDoIPPowerMode ()=0;`	
DIRECTION NOT DEFINED	void	–
Return value:	ara::core::Future< ara::diag::Power ModeType >	current diagnostic power mode
Header file:	#include "ara/diag/doip_power_mode.h"	
Description:	Called to get the current Power Mode for the DoIP protocol.	

⌋(*RS_AP_00138*, *SRS_Eth_00080*)

8.5.13.5 diag::DoIPPowerMode::Offer function

[SWS_DM_00735]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPPowerMode::Offer()	
Scope:	class ara::diag::DoIPPowerMode	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.

▽

△

Header file:	#include "ara/diag/doip_power_mode.h"
Description:	This Offer will enable the DM to forward request messages to this handler.

⌋(*RS_AP_00139*, *SRS_Eth_00080*)

8.5.13.6 diag::DoIPPowerMode::StopOffer function

[SWS_DM_00736]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DoIPPowerMode::StopOffer()
Scope:	class ara::diag::DoIPPowerMode
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/doip_power_mode.h"
Description:	This StopOffer will disable the forwaring of request messages from DM.

⌋(*SRS_Eth_00080*)

8.5.14 DoIPActivationLine class

The InstanceSpecifier is only compatible with PortInterface of `DiagnosticDoIPActivationLineInterface`. Note : For DoIPActivationLineInterface, DM has to have a R-PORT.

[SWS_DM_00830]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::DoIPActivationLine
Scope:	namespace ara::diag
Syntax:	`class DoIPActivationLine {...};`
Header file:	#include "ara/diag/doip_activationline.h"
Description:	DiagnosticDoIPActivationLineInterface.

⌋(*RS_Diag_04242*)

8.5.14.1 diag::DoIPActivationLine::DoIPActivationLine function

[SWS_DM_00831]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPActivationLine::DoIPActivationLine(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::diag::DoIPActivationLine	
Syntax:	`explicit DoIPActivationLine (const ara::core::InstanceSpecifier &specifier);`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an DiagnosticDoIPActivationLineInterface
Header file:	#include "ara/diag/doip_activationline.h"	
Description:	Constructor of DoIPActivationLine.	

⌋(*RS_Diag_04242*)

8.5.14.2 diag::DoIPActivationLine::~DoIPActivationLine function

[SWS_DM_00832]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DoIPActivationLine::~DoIPActivationLine()
Scope:	class ara::diag::DoIPActivationLine
Syntax:	`virtual ~DoIPActivationLine () noexcept=default;`
Exception Safety:	noexcept
Header file:	#include "ara/diag/doip_activationline.h"
Description:	Destructor of DoIPActivationLine.

⌋(*RS_Diag_04242*, *RS_AP_00134*)

8.5.14.3 diag::DoIPActivationLine::GetNetworkInterfaceId function

[SWS_DM_00833]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPActivationLine::GetNetworkInterfaceId()	
Scope:	class ara::diag::DoIPActivationLine	
Syntax:	`virtual ara::core::Future<std::uint8_t> GetNetworkInterfaceId ()=0;`	
DIRECTION NOT DEFINED	void	–
Return value:	ara::core::Future< std::uint8_t >	network interface id for which this activation line is responsible.
Header file:	#include "ara/diag/doip_activationline.h"	
Description:	Called to get the get the network interface Id (see DoIpNetworkConfiguration.networkInterface Id) for which this DoIPActivationLine instance is responsible.	

▽

△

Notes:	If the reported DoIpNetworkConfiguration.networkInterfaceId belongs to a DoIpNetwork Configuration with property isActivationLineDependent = 'FALSE', this is an error!

⌋(*RS_Diag_04242*)

8.5.14.4 diag::DoIPActivationLine::UpdateActivationLineState function

[SWS_DM_00834]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPActivationLine::UpdateActivationLineState(std::bool)	
Scope:	class ara::diag::DoIPActivationLine	
Syntax:	`virtual void UpdateActivationLineState (std::bool)=0;`	
DIRECTION NOT DEFINED	std::bool	–
Return value:	None	
Header file:	#include "ara/diag/doip_activationline.h"	
Description:	Called to update current activation line state.	

⌋(*RS_Diag_04242*)

8.5.14.5 diag::DoIPActivationLine::GetActivationLineState function

[SWS_DM_00835]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPActivationLine::GetActivationLineState()	
Scope:	class ara::diag::DoIPActivationLine	
Syntax:	`virtual ara::core::Future<std::bool> GetActivationLineState ()=0;`	
DIRECTION NOT DEFINED	void	–
Return value:	ara::core::Future< std::bool >	TRUE in case the activation line is active, elese FALSE.
Header file:	#include "ara/diag/doip_activationline.h"	
Description:	Called to get the current activation line state.	

⌋(*RS_Diag_04242*)

8.5.14.6 diag::DoIPActivationLine::Offer function

[SWS_DM_00836]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPActivationLine::Offer()	
Scope:	class ara::diag::DoIPActivationLine	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	–
Errors:	tbd	This error includes errors in offering this instance.
Header file:	#include "ara/diag/doip_activationline.h"	
Description:	This Offer will enable the DM to listen to activation line state changes for the given interface.	

⌋(*RS_Diag_04242*)

8.5.14.7 diag::DoIPActivationLine::StopOffer function

[SWS_DM_00837]{DRAFT} ⌈

Kind:	function
Symbol:	ara::diag::DoIPActivationLine::StopOffer()
Scope:	class ara::diag::DoIPActivationLine
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/diag/doip_activationline.h"
Description:	This StopOffer will disable the provision of activation line state to DM.

⌋(*RS_Diag_04242*)

8.5.15 DoIPTriggerVehicleAnnouncement class

For `DiagnosticDoIPTriggerVehicleAnnouncementInterface`, DM has to provide a P-Port per supported DoIP network interface.

[SWS_DM_00820]{DRAFT} ⌈

Kind:	class
Symbol:	ara::diag::DoIPTriggerVehicleAnnouncement
Scope:	namespace ara::diag
Syntax:	`class DoIPTriggerVehicleAnnouncement {...};`
Header file:	#include "ara/diag/doip_trigger_announcement.h"
Description:	DiagnosticDoIPTriggerVehicleAnnouncement.

⌋(*RS_Diag_04242*)

8.5.15.1 diag::DoIPTriggerVehicleAnnouncement::GetDoIPTriggerVehicleAnnouncement function

[SWS_DM_00821]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPTriggerVehicleAnnouncement::GetDoIPTriggerVehicleAnnouncement()	
Scope:	class ara::diag::DoIPTriggerVehicleAnnouncement	
Syntax:	`static Result<DoIPTriggerVehicleAnnouncement&> GetDoIPTriggerVehicleAnnouncement ();`	
Return value:	Result< DoIPTriggerVehicleAnnouncement & >	DoIPTriggerVehicleAnnouncement object
Header file:	#include "ara/diag/doip_trigger_announcement.h"	
Description:	Get DoIPTriggerVehicleAnnouncement interface from DM.	

⌋(*RS_Diag_04242*)

8.5.15.2 diag::DoIPTriggerVehicleAnnouncement::TriggerVehicleAnnouncement function

[SWS_DM_00822]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::diag::DoIPTriggerVehicleAnnouncement::TriggerVehicleAnnouncement(std::uint8_t networkInterfaceId)	
Scope:	class ara::diag::DoIPTriggerVehicleAnnouncement	
Syntax:	`void TriggerVehicleAnnouncement (std::uint8_t networkInterfaceId)=0;`	
DIRECTION NOT DEFINED	networkInterfaceId	–
Return value:	None	
Header file:	#include "ara/diag/doip_trigger_announcement.h"	
Description:	Called by application to frigger DM sending out vehicle announcements on the given network interface Id.	
Notes:	If the reported DoIpNetworkConfiguration.networkInterfaceId belongs to a DoIpNetwork Configuration with property isActivationLineDependent = 'TRUE', this is an error as on those interfaces sending of announcements happens automatically after activation line going up/ip address assignment.	

⌋(*RS_Diag_04242*)

A Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**ApApplicationError**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to formally specify the semantics of an application error on the AUTOSAR adaptive platform **Tags:** atp.Status=draft atp.recommendedPackage=ApplicationErrors			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
errorCode	Integer	1	attr	This attribute has the ability to specify the error code value within the enclosing AdaptivePlatformApplication Error.
errorDomain	ApApplicationError Domain	1	ref	This reference represents the error domain of the Ap ApplicationError. **Tags:**atp.Status=draft

Table A.1: ApApplicationError

Class	***CppImplementationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a reusable data type definition taken as a the basis for a C++ language binding **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	CustomCppImplementationDataType, StdCppImplementationDataType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arraySize	PositiveInteger	0..1	attr	This attribute can be used to specify the array size if the enclosing CppImplementationDataType has array semantics. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CppImplementationData Type. **Tags:**atp.Status=draft
subElement (ordered)	CppImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CppImplementationDataType **Tags:**atp.Status=draft
template Argument (ordered)	CppTemplateArgument	*	aggr	This aggreation allows for the specification of properties of template arguments **Tags:**atp.Status=draft
typeEmitter	NameToken	0..1	attr	This attribute can be taken to control how the respective CppImplementationDataType is contributed to the language binding.

△

Class	***CppImplementationDataType*** (abstract)			
typeReference	CppImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef). **Tags:**atp.Status=draft

Table A.2: CppImplementationDataType

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***	***Mult.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level.

Table A.3: DataPrototype

Class	**DiagEventDebounceCounterBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the counter-based debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemDebounceAlgorithmClass to DemDebounce CounterBased.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
counterBased FdcThreshold StorageValue	Integer	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.
counter DecrementStep Size	Integer	1	attr	This value shall be taken to decrement the internal debounce counter. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterFailed Threshold	Integer	1	attr	This value defines the event-specific limit that indicates the "failed" counter status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counter IncrementStep Size	Integer	1	attr	This value shall be taken to increment the internal debounce counter. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterJump Down	Boolean	1	attr	This value activates or deactivates the counter jump-down behavior. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

▽

△

Class	**DiagEventDebounceCounterBased**			
counterJump DownValue	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from incrementing to decrementing. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterJumpUp	Boolean	1	attr	This value activates or deactivates the counter jump-up behavior. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterJumpUp Value	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from decrementing to incrementing. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterPassed Threshold	Integer	1	attr	This value defines the event-specific limit that indicates the "passed" counter status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table A.4: DiagEventDebounceCounterBased

Class	**DiagEventDebounceTimeBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the time-based pre-debounce algorithm shall be used by the Dem for this diagnostic monitor. This is related to set the EcuC choice container DemDebounceAlgorithmClass to DemDebounceTime Base.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
timeBasedFdc Threshold StorageValue	TimeValue	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
timeFailed Threshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "failed" status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
timePassed Threshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "passed" status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild

Table A.5: DiagEventDebounceTimeBased

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

Table A.6: DiagnosticAbstractDataIdentifier

Class	**DiagnosticAccessPermission**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm			
Note	This represents the specification of whether a given service can be accessed according to the existence of meta-classes referenced by a particular DiagnosticAccessPermission. In other words, this meta-class acts as a mapping element between several (otherwise unrelated) pieces of information that are put into context for the purpose of checking for access rights. **Tags:**atp.recommendedPackage=DiagnosticAccessPermissions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
diagnostic Session	DiagnosticSession	*	ref	This represents the associated DiagnosticSessions
environmental Condition	Diagnostic EnvironmentalCondition	0..1	ref	This represents the environmental conditions associated with the access permission.
securityLevel	DiagnosticSecurityLevel	*	ref	This represents the associated DiagnosticSecurityLevels

Table A.7: DiagnosticAccessPermission

Class	**DiagnosticAging**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticAging			
Note	Defines the aging algorithm. **Tags:**atp.recommendedPackage=DiagnosticAgings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
agingCycle	DiagnosticOperation Cycle	0..1	ref	This represents the applicable aging cycle. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=agingCycle, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**DiagnosticAging**			
threshold	PositiveInteger	0..1	attr	Number of aging cycles needed to unlearn/delete the event. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table A.8: DiagnosticAging

Class	**DiagnosticClearCondition**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::DiagnosticDesign::DiagnosticClearCondition			
Note	This meta-class describes a clear condition for diagnostic purposes. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticConditions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticCondition*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.9: DiagnosticClearCondition

Enumeration	**DiagnosticClearDtcLimitationEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticCommonProps
Note	Scope of the DEM_ClearDTC Api.
Literal	***Description***
allSupportedDtcs	DEM_ClearDtc API accepts all supported DTC values. **Tags:**atp.EnumerationLiteralIndex=0
clearAllDtcs	DEM_ClearDtc API accepts ClearAllDTCs only. **Tags:**atp.EnumerationLiteralIndex=1

Table A.10: 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.EnumerationLiteralIndex=0
onlyThisCycleAnd Readiness	The OperationCycle and readiness bits of the event status byte are reset. **Tags:**atp.EnumerationLiteralIndex=1

Table A.11: 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***	***Mult.***	***Kind***	***Note***
comControl Class	DiagnosticComControl Class	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticComControl in the given context.
customSub Function Number	PositiveInteger	0..1	attr	This attribute shall be used to define a custom sub-function number if none of the standardized values of category shall be used.

Table A.12: 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***	***Mult.***	***Kind***	***Note***
agingRequires TestedCycle	Boolean	1	attr	Defines whether the aging cycle counter is processed every aging cycles or else only tested aging cycle are considered. If the attribute is set to TRUE: only tested aging cycle are considered for aging cycle counter. If the attribute is set to FALSE: aging cycle counter is processed every aging cycle.
clearDtc Limitation	DiagnosticClearDtc LimitationEnum	1	attr	Defines the scope of the DEM_ClearDTC Api.
debounce AlgorithmProps	DiagnosticDebounce AlgorithmProps	*	aggr	Defines the used debounce algorithms relevant in the context of the enclosing DiagnosticCommonProps. Usually, there is a variety of debouncing algorithms to take into account and therefore the multiplicity of this aggregation is set to 0..*.
default Endianness	ByteOrderEnum	1	attr	Defines the default endianness of the data belonging to a DID or RID which is applicable if the DiagnosticData Element does not define the endianness via the swData DefProps.baseType attribute.
environment DataCapture	DiagnosticDataCapture Enum	0..1	attr	This attribute determines whether the capturing of environment data is done synchronously inside the report API function or whether the capturing shall be done asynchronously, i.e. after the report API function already terminated.
event Displacement Strategy	DiagnosticEvent DisplacementStrategy Enum	1	attr	This attribute defines, whether support for event displacement is enabled or not, and which displacement strategy is followed.
maxNumberOf EventEntries	PositiveInteger	0..1	attr	This attribute fixes the maximum number of event entries in the fault memory.

△

Class	<<atpVariation>> **DiagnosticCommonProps**			
maxNumberOf Request Correctly Received Response Pending	PositiveInteger	1	attr	Maximum number of negative responses with response code 0x78 (requestCorrectlyReceived-ResponsePending) allowed per request. DCM will send a negative response with response code 0x10 (generalReject), in case the limit value gets reached. Value 0xFF means that no limit number of NRC 0x78 response apply.
memoryEntry StorageTrigger	DiagnosticMemoryEntry StorageTriggerEnum	1	attr	Describes the primary trigger to allocate an event memory entry.
occurrence Counter Processing	DiagnosticOccurrence CounterProcessing Enum	1	attr	This attribute defines the consideration of the fault confirmation process for the occurrence counter.
resetConfirmed BitOnOverflow	Boolean	1	attr	This attribute defines, whether the confirmed bit is reset or not while an event memory entry will be displaced.
responseOnAll RequestSids	Boolean	1	attr	If set to FALSE the DCM will not respond to diagnostic request that contains a service ID which is in the range from 0x40 to 0x7F or in the range from 0xC0 to 0xFF (Response IDs).
responseOn Second Declined Request	Boolean	1	attr	Defines the reaction upon a second request (ClientB) that can not be processed (e.g. due to priority assessment). TRUE: when the second request (Client B) can not be processed, it shall be answered with NRC21 BusyRepeat Request. FALSE: when the second request (Client B) can not be processed, it shall not be responded.
securityDelay TimeOnBoot	TimeValue	1	attr	Start delay timer on power on in seconds. This delay indicates the time at ECU boot power-on time where the Dcm remains in the default session and does not accept a security access.
statusBit HandlingTest FailedSinceLast Clear	DiagnosticStatusBit HandlingTestFailed SinceLastClearEnum	1	attr	This attribute defines, whether the aging and displacement mechanism shall be applied to the "Test FailedSinceLastClear" status bits.
statusBit StorageTest Failed	Boolean	1	attr	This parameter is used to activate/deactivate the permanent storage of the "TestFailed" status bits. true: storage activated false: storage deactivated
typeOfDtc Supported	DiagnosticTypeOfDtc SupportedEnum	1	attr	This attribute defines the format returned by Dem_Dcm GetTranslationType and does not relate to/influence the supported Dem functionality.

Table A.13: DiagnosticCommonProps

Class	**DiagnosticConditionInterface**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface
Note	This meta-class represents the ability to implement a PortInterface to process requests for diagnostic conditions on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces

▽

△

Class	DiagnosticConditionInterface			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.14: DiagnosticConditionInterface

Class	DiagnosticConnectedIndicator			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticEvent			
Note	Description of indicators that are defined per DiagnosticEvent.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
behavior	DiagnosticConnected IndicatorBehaviorEnum	0..1	attr	Behavior of the linked indicator.
healingCycle	DiagnosticOperation Cycle	1	ref	The deactivation of indicators per event is defined as healing of a diagnostic event. The operation cycle in which the warning indicator will be switched off is defined here.
healingCycle Counter Threshold	PositiveInteger	0..1	attr	This attribute defines the number of healing cycles for the WarningIndicatorOffCriteria **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
indicator	DiagnosticIndicator	1	ref	Reference to the used indicator.

Table A.15: DiagnosticConnectedIndicator

Class	DiagnosticContributionSet			
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticContribution			
Note	This meta-class represents a root node of a diagnostic extract. It bundles a given set of diagnostic model elements. The granularity of the DiagonsticContributionSet is arbitrary in order to support the aspect of decentralized configuration, i.e. different contributors can come up with an own DiagnosticContribution Set. **Tags:**atp.recommendedPackage=DiagnosticContributionSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
common Properties	DiagnosticCommon Props	0..1	aggr	This attribute represents a collection of diagnostic properties that are shared among the entire Diagnostic ContributionSet. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=commonProperties, variation Point.shortLabel

▽

△

Class	**DiagnosticContributionSet**			
element	DiagnosticCommon Element	*	ref	This represents a DiagnosticCommonElement considered in the context of the DiagnosticContributionSet **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element, variationPoint.shortLabel vh.latestBindingTime=postBuild
serviceTable	DiagnosticServiceTable	*	ref	This represents the collection of DiagnosticServiceTables to be considered in the scope of this Diagnostic ContributionSet. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceTable, variationPoint.shortLabel vh.latestBindingTime=postBuild

Table A.16: DiagnosticContributionSet

Class	**DiagnosticControlDTCSetting**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::ControlDTCSetting			
Note	This represents an instance of the "Control DTC Setting" diagnostic service. **Tags:**atp.recommendedPackage=DiagnosticControlDtcSettings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
dtcSettingClass	DiagnosticControlDTC SettingClass	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 DiagnosticControlDTCSetting in the given context.
dtcSetting Parameter	PositiveInteger	1	attr	This represents the DTCSettingType defined by ISO 14229-1. The pre-defined values are 1 (ON) and 2 (OFF).

Table A.17: DiagnosticControlDTCSetting

Class	**DiagnosticCustomServiceInstance**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::CustomServiceInstance			
Note	This meta-class has the ability to define an instance of a custom diagnostic service. **Tags:**atp.recommendedPackage=DiagnosticCustomInstances			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
customService Class	DiagnosticCustom ServiceClass	0..1	ref	Reference to the corresponding DiagnosticCustom ServiceClass.

Table A.18: DiagnosticCustomServiceInstance

Class	**DiagnosticDTCInformationInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to access the properties of DTCs on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.19: DiagnosticDTCInformationInterface

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

Table A.20: 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***	***Mult.***	***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.
scalingInfoSize	PositiveInteger	0..1	attr	Size in bytes of scaling information for the DiagnosticDataElement if used with DiagnosticReadScalingDataByIdentifier
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.21: DiagnosticDataElement

Class	DiagnosticDataElementInterface			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a element-of-DID-focused PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticAbstractDataIdentifierInterface*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
read	ClientServerOperation	0..1	aggr	This represents the method to read the content of an element of a diagnostic data identifier. **Tags:**atp.Status=draft
write	ClientServerOperation	0..1	aggr	This represents the method to write the content of an element of a diagnostic data identifier. **Tags:**atp.Status=draft

Table A.22: DiagnosticDataElementInterface

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**	**Mult.**	**Kind**	**Note**
dataElement	DiagnosticParameter	1..*	aggr	This is the dataElement associated with the Diagnostic DataIdentifier. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bitOffset, variationPoint.shortLabel vh.latestBindingTime=postBuild
didSize	PositiveInteger	0..1	attr	This attribute indicates the size in bytes of the Diagnostic DataIdentifier.
representsVin	Boolean	0..1	attr	This attributes indicates whether the specific Diagnostic DataIdentifier represents the vehicle identification.
supportInfoByte	DiagnosticSupportInfoByte	0..1	aggr	This attribute represents the supported information associated with the DiagnosticDataIdentifier.

Table A.23: DiagnosticDataIdentifier

Class	DiagnosticDataIdentifierGenericInterface
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface

▽

△

Class	**DiagnosticDataIdentifierGenericInterface**			
Note	This meta-class represents the ability to implement a generic DID-focused PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticAbstractDataIdentifierInterface*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.24: DiagnosticDataIdentifierGenericInterface

Class	**DiagnosticDataIdentifierInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a DID-focused PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticAbstractDataIdentifierInterface*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
read	ClientServerOperation	0..1	aggr	This represents the method to read the content of a diagnostic data identifier. **Tags:**atp.Status=draft
write	ClientServerOperation	0..1	aggr	This represents the method to write the contents of a diagnostic data identifier. **Tags:**atp.Status=draft

Table A.25: DiagnosticDataIdentifierInterface

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

Table A.26: DiagnosticDataIdentifierSet

Class	**DiagnosticDebounceAlgorithmProps**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticDebouncingAlgorithm			
Note	Defines properties for the debounce algorithm class.			
Base	*ARObject*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
debounce Algorithm	DiagEventDebounce Algorithm	1	aggr	This represents the actual debounce algorithm.
debounce Behavior	DiagnosticDebounce BehaviorEnum	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. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
debounce CounterStorage	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.27: DiagnosticDebounceAlgorithmProps

Enumeration	**DiagnosticDebounceBehaviorEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticDebouncingAlgorithm
Note	Event debounce algorithm behavior options.
Literal	***Description***
freeze	The event debounce counter will be frozen with the current value and will not change while a related enable condition is not fulfilled or ControlDTCSetting of the related event is disabled. After all related enable conditions are fulfilled and ControlDTCSetting of the related event is enabled again, the event qualification will continue with the next report of the event (i.e. SetEventStatus). **Tags:**atp.EnumerationLiteralIndex=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.EnumerationLiteralIndex=1

Table A.28: DiagnosticDebounceBehaviorEnum

Class	**DiagnosticDoIPActivationLineInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to implement the DoIPActivationLine on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.29: DiagnosticDoIPActivationLineInterface

Class	**DiagnosticDoIPGroupIdentificationInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to implement the DoIP Group Identification on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.30: DiagnosticDoIPGroupIdentificationInterface

Class	**DiagnosticDoIPPowerModeInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to implement the DoIP Power Mode on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.31: DiagnosticDoIPPowerModeInterface

Class	**DiagnosticDoIPTriggerVehicleAnnouncementInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to implement the DoIPTriggerVehicle Announcement on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.32: DiagnosticDoIPTriggerVehicleAnnouncementInterface

Class	**DiagnosticDownloadInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to process requests for downloading data using diagnostic channels on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.33: DiagnosticDownloadInterface

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***	***Mult.***	***Kind***	***Note***
customSub Function Number	PositiveInteger	0..1	attr	This attribute shall be used to define a custom sub-function number if none of the standardized values of category shall be used.
ecuResetClass	DiagnosticEcuReset Class	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticEcuReset in the given context.

Table A.34: 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***	***Mult.***	***Kind***	***Note***
respondTo Reset	DiagnosticResponseTo EcuResetEnum	0..1	attr	This attribute defines whether the response to the Ecu Reset service shall be transmitted before or after the actual reset.

Table A.35: 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*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.36: 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*	*Mult.*	*Kind*	*Note*
compareType	DiagnosticCompare TypeEnum	1	attr	This attributes represents the concrete type of the comparison.

Table A.37: DiagnosticEnvCompareCondition

Class	**DiagnosticEnvConditionFormula**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EnvironmentalCondition			
Note	A DiagnosticEnvConditionFormula embodies the computation instruction that is to be evaluated at runtime to determine if the DiagnosticEnvironmentalCondition is currently present (i.e. the formula is evaluated to true) or not (otherwise). The formula itself consists of parts which are combined by the logical operations specified by DiagnosticEnvConditionFormula.op. If a diagnostic functionality cannot be executed because an environmental condition fails then the diagnostic stack shall send a negative response code (NRC) back to the client. The value of the NRC is directly related to the specific formula and is therefore formalized in the attribute DiagnosticEnvCondition Formula.nrcValue.			
Base	*ARObject*, *DiagnosticEnvConditionFormulaPart*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
nrcValue	PositiveInteger	0..1	attr	This attribute represents the concrete NRC value that shall be returned if the condition fails.
op	DiagnosticLogical OperatorEnum	1	attr	This attribute represents the concrete operator (supported operators: and, or) of the condition formula.
part (ordered)	DiagnosticEnvCondition FormulaPart	*	aggr	This aggregation represents the collection of formula parts that can be combined by logical operators.

Table A.38: 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***	***Mult.***	***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.39: DiagnosticEnvDataCondition

Class	**DiagnosticEnvironmentalCondition**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::EnvironmentalCondition			
Note	The meta-class DiagnosticEnvironmentalCondition 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***	***Mult.***	***Kind***	***Note***
formula	DiagnosticEnvCondition Formula	1	aggr	This attribute represents the formula part of the DiagnosticEnvironmentalCondition.
modeElement	DiagnosticEnvMode Element	*	aggr	This aggregation contains a representation of Mode Declarations in the context of a DiagnosticEnvironmental Condition.

Table A.40: 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***	***Mult.***	***Kind***	***Note***
clearEvent Behavior	DiagnosticClearEvent BehaviorEnum	0..1	attr	This attribute defines the resulting UDS DTC status byte for the related event, which shall not be cleared according to the ClearEventAllowed callback.
connected Indicator	DiagnosticConnected Indicator	*	aggr	Event specific description of Indicators. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
eventClear Allowed	DiagnosticEventClear AllowedEnum	0..1	attr	This attribute defines whether the Dem has access to a "ClearEventAllowed" callback.

▽

△

Class	**DiagnosticEvent**			
eventFailure CycleCounter Threshold	PositiveInteger	0..1	attr	This attribute defines the number of failure cycles for the event based fault confirmation. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
prestorage FreezeFrame	Boolean	1	attr	This attribute describes whether the Prestorage of Freeze Frames is supported by the assigned event or not. True: Prestorage of FreezeFrames is supported False: Prestorage of FreezeFrames is not supported
prestored Freezeframe StoredInNvm	Boolean	0..1	attr	If the Event uses a prestored freeze-frame (using the operations PrestoreFreezeFrame and ClearPrestored FreezeFrame of the service interface DiagnosticMonitor) this attribute indicates if the Event requires the data to be stored in non-volatile memory. TRUE = Dem shall store the prestored data in non-volatile memory, FALSE = Data can be lost at shutdown (not stored in Nvm)
recoverableIn SameOperation Cycle	Boolean	0..1	attr	If the attribute is set to true then reporting PASSED will reset the indication of a failed test in the current operation cycle. If the attribute is set to false then reporting PASSED will be ignored and not lead to a reset of the indication of a failed test.

Table A.41: DiagnosticEvent

Class	**DiagnosticEventInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to access the properties of diagnostic events on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.42: DiagnosticEventInterface

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***	***Mult.***	***Kind***	***Note***
debounce Algorithm	DiagnosticDebounce AlgorithmProps	1	ref	Reference to a DebounceAlgorithm assigned to a DiagnosticEvent.

▽

△

Class	**DiagnosticEventToDebounceAlgorithmMapping**			
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which a Debounce Algorithm is assigned.

Table A.43: 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***	***Mult.***	***Kind***	***Note***
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which an Enable ConditionGroup is assigned.
enableCondition Group	DiagnosticEnable ConditionGroup	1	ref	Reference to an EnableConditionGroup assigned to a DiagnosticEvent.

Table A.44: 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***	***Mult.***	***Kind***	***Note***
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which an Operation Cycle is assigned.
operationCycle	DiagnosticOperation Cycle	1	ref	Reference to an OperationCycle assigned to a Diagnostic Event.

Table A.45: DiagnosticEventToOperationCycleMapping

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***	***Mult.***	***Kind***	***Note***
diagnosticEvent	DiagnosticEvent	1	ref	Reference to a DiagnosticEvent to which a UDS Diagnostic Trouble Code is assigned.

▽

△

Class	**DiagnosticEventToTroubleCodeUdsMapping**			
troubleCodeUds	DiagnosticTroubleCodeUds	1	ref	Reference to an UDS Diagnostic Trouble Code assigned to a DiagnosticEvent.

Table A.46: DiagnosticEventToTroubleCodeUdsMapping

Class	**DiagnosticExtendedDataRecord**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticExtendedDataRecord			
Note	Description of an extended data record. **Tags:**atp.recommendedPackage=DiagnosticExtendedDataRecords			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
customTrigger	String	0..1	attr	This attribute shall be taken to verbally describe the nature of the custom trigger.
recordElement	DiagnosticParameter	*	aggr	Defined DataElements in the extended record element.
recordNumber	PositiveInteger	1	attr	This attribute specifies an unique identifier for an extended data record.
trigger	DiagnosticRecordTriggerEnum	1	attr	This attribute specifies the primary trigger to allocate an event memory entry.
update	Boolean	1	attr	This attribute defines when an extended data record is captured. True: This extended data record is captured every time. False: This extended data record is only captured for new event memory entries.

Table A.47: 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*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
customTrigger	String	0..1	attr	This attribute shall be taken to verbally describe the nature of the custom trigger.
recordNumber	PositiveInteger	0..1	attr	This attribute defines a record number for a freeze frame record. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
trigger	DiagnosticRecordTriggerEnum	1	attr	This attribute defines the primary trigger to allocate an event memory entry.

▽

△

Class	DiagnosticFreezeFrame			
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.48: DiagnosticFreezeFrame

Class	DiagnosticGenericUdsInterface			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a generic UDS PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.49: DiagnosticGenericUdsInterface

Class	DiagnosticIndicatorInterface			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to implement indicator functionality on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.50: DiagnosticIndicatorInterface

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*

▽

△

Class	**DiagnosticMapping** (abstract)			
Subclasses	DiagnosticEventToDebounceAlgorithmMapping, DiagnosticEventToEnableConditionGroupMapping, DiagnosticEventToOperationCycleMapping, DiagnosticEventToTroubleCodeJ1939Mapping, DiagnosticEventToTroubleCodeUdsMapping, DiagnosticFimAliasEventGroupMapping, DiagnosticFimAliasEventMapping, DiagnosticInhibitSourceEventMapping, DiagnosticJ1939SpnMapping, DiagnosticProvidedDataMapping, DiagnosticServiceDataMapping, *DiagnosticSwMapping*, DiagnosticTroubleCodeUdsToClearConditionGroupMapping, DiagnosticTroubleCodeUdsToTroubleCodeObdMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.51: 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**	**Mult.**	**Kind**	**Note**
dtcStatusAvailabilityMask	PositiveInteger	0..1	attr	Mask for the supported DTC status bits by the Dem.
typeOfFreezeFrameRecordNumeration	DiagnosticTypeOfFreezeFrameRecordNumerationEnum	0..1	attr	This attribute defines the type of assigning freeze frame record numbers for event-specific freeze frame records.

Table A.52: DiagnosticMemoryDestination

Class	**DiagnosticMemoryDestinationPrimary**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This represents a primary memory for a diagnostic event. **Tags:**atp.recommendedPackage=DiagnosticMemoryDestinations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMemoryDestination*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.53: 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**	**Mult.**	**Kind**	**Note**

▽

△

Class	**DiagnosticMemoryDestinationUserDefined**			
memoryId	PositiveInteger	1	attr	This represents the identifier of the user-defined memory.

Table A.54: DiagnosticMemoryDestinationUserDefined

Class	**DiagnosticMonitorInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a monitor-focused PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.55: DiagnosticMonitorInterface

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***	***Mult.***	***Kind***	***Note***
automaticEnd	Boolean	1	attr	If set to true the driving cycle shall automatically end at either Dem_Shutdown() or Dem_Init().
cycleAutostart	Boolean	1	attr	This attribute defines if the operation cycles is automatically re-started during Dem_PreInit.
cycleStatus Storage	Boolean	1	attr	Defines if the operation cycle state is available over the power cycle (stored non-volatile) or not. • true: the operation cycle state is stored non-volatile • false: the operation cycle state is only stored volatile
type	DiagnosticOperation CycleTypeEnum	1	attr	Operation cycles types for the Dem.

Table A.56: DiagnosticOperationCycle

Class	**DiagnosticOperationCycleInterface**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface

▽

△

Class	DiagnosticOperationCycleInterface			
Note	This meta-class represents the ability to implement a PortInterface to process requests for operation cycles on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.57: DiagnosticOperationCycleInterface

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**	**Mult.**	**Kind**	**Note**
bitOffset	PositiveInteger	1	attr	This represents the bitOffset of the DiagnosticParameter
dataElement	DiagnosticDataElement	1	aggr	This represents the related dataElement of the Diagnostic Parameter **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
supportInfo	DiagnosticParameter SupportInfo	0..1	aggr	This attribute represents the ability to define which bit of the support info byte is representing this part of the PID.

Table A.58: DiagnosticParameter

Class	***DiagnosticPortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class serves as an abstract base-class for all diagnostics-related PortInterfaces. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Subclasses	*DiagnosticAbstractDataIdentifierInterface*, *DiagnosticAbstractRoutineInterface*, DiagnosticConditionInterface, DiagnosticDTCInformationInterface, DiagnosticDoIPActivationLineInterface, DiagnosticDoIPGroupIdentificationInterface, DiagnosticDoIPPowerModeInterface, DiagnosticDoIPTriggerVehicleAnnouncementInterface, DiagnosticDownloadInterface, DiagnosticEventInterface, DiagnosticGenericUdsInterface, DiagnosticIndicatorInterface, DiagnosticMonitorInterface, DiagnosticOperationCycleInterface, DiagnosticSecurityLevelInterface, DiagnosticServiceValidationInterface, DiagnosticUploadInterface			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.59: DiagnosticPortInterface

Class	**DiagnosticProvidedDataMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::DiagnosticDesign::DiagnosticProvidedDataMapping			
Note	This represents the ability to define the nature of a data access for a DiagnosticDataElement based on a data provider that cannot be modeled explicitly. **Tags:** atp.Status=draft atp.recommendedPackage=DataMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataElement	DiagnosticDataElement	0..1	ref	This represents the DiagnosticDataElement for which the access is further qualified by the DiagnosticProvidedData Mapping.dataProvider. **Tags:**atp.Status=draft
dataProvider	NameToken	1	attr	This represents the ability to further specify the data provider.

Table A.60: DiagnosticProvidedDataMapping

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***	***Mult.***	***Kind***	***Note***
read DTCInformation Class	DiagnosticReadDTC InformationClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticRead DTCInformation in the given context.

Table A.61: 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***	***Mult.***	***Kind***	***Note***
readClass	DiagnosticReadDataBy IdentifierClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticReadDataBy Identifier in the given context.

Table A.62: DiagnosticReadDataByIdentifier

Class	**DiagnosticReadDataByIdentifierClass**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::DataByIdentifier			
Note	This meta-class contains attributes shared by all instances of the "Read Data by Identifier" diagnostic service. **Tags:**atp.recommendedPackage=DiagnosticDataByIdentifiers			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceClass*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
maxDidToRead	PositiveInteger	1	attr	This attribute represents the maximum number of allowed DIDs in a single instance of DiagnosticReadDataBy Identifier.

Table A.63: 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.EnumerationLiteralIndex=0
respondBefore Reset	Answer to EcuReset service should come before the reset. **Tags:**atp.EnumerationLiteralIndex=1

Table A.64: 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*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
id	PositiveInteger	1	attr	This is the numerical identifier used to identify the DiagnosticRoutine in the scope of diagnostic workflow **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
requestResult	DiagnosticRequest RoutineResults	0..1	aggr	This represents the ability to request the result of a running routine.
routineInfo	PositiveInteger	0..1	attr	This represents the routine info byte. The info byte contains a manufacturer-specific value (for the identification of record identifiers) that is reported to the tester. Other use cases for this attribute are mentioned in ISO 27145 and ISO 26021.
start	DiagnosticStartRoutine	0..1	aggr	This represents the ability to start a routine
stop	DiagnosticStopRoutine	0..1	aggr	This represents the ability to stop a running routine.

Table A.65: DiagnosticRoutine

Class	**DiagnosticRoutineControl**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::RoutineControl			
Note	This represents an instance of the "Routine Control" diagnostic service. **Tags:**atp.recommendedPackage=DiagnosticRoutineControls			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
routine	DiagnosticRoutine	1	ref	This refers to the applicable DiagnosticRoutine.
routineControl Class	DiagnosticRoutine 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 DiagnosticRoutineControl in the given context.

Table A.66: DiagnosticRoutineControl

Class	**DiagnosticRoutineGenericInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a generic Routine-focused PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticAbstractRoutineInterface*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.67: DiagnosticRoutineGenericInterface

Class	**DiagnosticRoutineInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a routine-focused PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticAbstractRoutineInterface*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
requestResult	ClientServerOperation	0..1	aggr	This represents the request result method of the diagnostic routine. **Tags:**atp.Status=draft
start	ClientServerOperation	0..1	aggr	This represents the start method of the diagnostic routine. **Tags:**atp.Status=draft

▽

△

Class	DiagnosticRoutineInterface			
stop	ClientServerOperation	0..1	aggr	This represents the stop method of the diagnostic routine. **Tags:**atp.Status=draft

Table A.68: DiagnosticRoutineInterface

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**	**Mult.**	**Kind**	**Note**
access Permission	DiagnosticAccess Permission	0..1	ref	This reference represents the access permission of the owning routine subfunction.

Table A.69: 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**	**Mult.**	**Kind**	**Note**
requestSeedId	PositiveInteger	1	attr	This would be 0x01, 0x03, 0x05, ... The sendKey id can be computed by adding 1 to the requestSeedId
securityAccess Class	DiagnosticSecurity AccessClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticSecurityAccess in the given context.
securityLevel	DiagnosticSecurityLevel	1	ref	This reference identifies the applicable security level for the security access. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=securityLevel

Table A.70: DiagnosticSecurityAccess

Class	DiagnosticSecurityAccessClass
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::SecurityAccess
Note	This meta-class contains attributes shared by all instances of the "Security Access" diagnostic service. **Tags:**atp.recommendedPackage=DiagnosticSecurityAccesss
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticServiceClass*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*

▽

△

Class	**DiagnosticSecurityAccessClass**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sharedTimer	Boolean	0..1	attr	Switch between separate or single shared timer instance and timer value. • True: use shared timer instance and timer value for all security access levels combined. • False: use separate timer instance and timer values for each security level. **Tags:**atp.Status=draft

Table A.71: DiagnosticSecurityAccessClass

Class	**DiagnosticSecurityLevel**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm			
Note	This meta-class represents the ability to define a security level considered for diagnostic purposes. **Tags:**atp.recommendedPackage=DiagnosticSecurityLevels			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
accessData RecordSize	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.
numFailed SecurityAccess	PositiveInteger	0..1	attr	This represents the number of failed security accesses after which the delay time is activated.
securityDelay Time	TimeValue	1	attr	This represents the delay time after a failed security access. Unit: second.
seedSize	PositiveInteger	1	attr	This represents the size of the security seed. Unit: byte.

Table A.72: DiagnosticSecurityLevel

Class	**DiagnosticSecurityLevelInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a security-level-focused PortInterface for diagnostics on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.73: DiagnosticSecurityLevelInterface

Class	***DiagnosticServiceClass*** (abstract)			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::CommonService			
Note	This meta-class provides the ability to define common properties that are shared among all instances of sub-classes of DiagnosticServiceInstance.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	DiagnosticClearDiagnosticInformationClass, DiagnosticClearResetEmissionRelatedInfoClass, DiagnosticComControlClass, DiagnosticControlDTCSettingClass, DiagnosticCustomServiceClass, DiagnosticDataTransferClass, DiagnosticDynamicallyDefineDataIdentifierClass, DiagnosticEcuResetClass, DiagnosticIoControlClass, DiagnosticReadDTCInformationClass, DiagnosticReadDataByIdentifierClass, DiagnosticReadDataByPeriodicIDClass, DiagnosticReadMemoryByAddressClass, DiagnosticReadScalingDataByIdentifierClass, DiagnosticRequestControlOfOnBoardDeviceClass, DiagnosticRequestCurrentPowertrainDataClass, DiagnosticRequestDownloadClass, DiagnosticRequestEmissionRelatedDTCClass, DiagnosticRequestEmissionRelatedDTCPermanentStatusClass, DiagnosticRequestFileTransferClass, DiagnosticRequestOnBoardMonitoringTestResultsClass, DiagnosticRequestPowertrainFreezeFrameDataClass, DiagnosticRequestUploadClass, DiagnosticRequestVehicleInfoClass, DiagnosticResponseOnEventClass, DiagnosticRoutineControlClass, DiagnosticSecurityAccessClass, DiagnosticSessionControlClass, DiagnosticTransferExitClass, DiagnosticWriteDataByIdentifierClass, DiagnosticWriteMemoryByAddressClass			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
access Permission	DiagnosticAccessPermission	0..1	ref	This represents the collection of DiagnosticAccess Permissions that allow for the execution of the referencing DiagnosticServiceClass.
access Permission Validity	DiagnosticAccessPermissionValidityEnum	1	attr	This attribute is responsible for clarifying the validity of the accessPermission reference.

Table A.74: DiagnosticServiceClass

Class	**DiagnosticServiceDataIdentifierPortMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::DiagnosticDesign::DiagnosticMapping			
Note	This meta-class provides the ability to define a diagnostic access to an entire DID. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *DiagnosticSwMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
diagnosticData Identifier	DiagnosticDataIdentifier	0..1	ref	This reference represents the applicable DiagnosticData Identfiier. **Tags:**atp.Status=draft
process	ProcessDesign	1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft

▽

△

Class	**DiagnosticServiceDataIdentifierPortMapping**			
swcService DependencyIn Executable	SwcService Dependency	0..1	iref	This reference identifies the applicable SwcService Dependency. The reference has the ability to point into the component hierarchy (under possible consideration of the rootSoftwareComposition). **Tags:**atp.Status=draft

Table A.75: DiagnosticServiceDataIdentifierPortMapping

Class	**DiagnosticServiceDataMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::ServiceMapping			
Note	This represents the ability to define a mapping of a diagnostic service to a software-component. This kind of service mapping is applicable for the usage of SenderReceiverInterfaces or event/notifier semantics in ServiceInterfaces on the adaptive platform. **Tags:**atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
diagnosticData Element	DiagnosticDataElement	1	ref	This represents the applicable payload that corresponds to the referenced DataPrototype in the role mappedData Element or (in case of a usage on the adaptive platform) mappedApDataElement.
mappedApData Element	DataPrototype	0..1	iref	This represents the dataElement in the application software of an adaptive AUTOSAR application that is accessed for diagnostic purpose. **Tags:**atp.Status=draft
mappedData Element	DataPrototype	0..1	iref	This represents the dataElement in the application software that is accessed for diagnostic purpose. This role is applicable on the classic platform.
process	ProcessDesign	0..1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft

Table A.76: DiagnosticServiceDataMapping

Class	***DiagnosticServiceInstance*** (abstract)
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm::DiagnosticService::CommonService
Note	This represents a concrete instance of a diagnostic service.
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*
Subclasses	DiagnosticClearDiagnosticInformation, DiagnosticClearResetEmissionRelatedInfo, DiagnosticCom Control, DiagnosticControlDTCSetting, DiagnosticCustomServiceInstance, *DiagnosticDataByIdentifier*, DiagnosticDynamicallyDefineDataIdentifier, DiagnosticEcuReset, DiagnosticIOControl, *Diagnostic MemoryByAddress*, DiagnosticReadDTCInformation, DiagnosticReadDataByPeriodicID, Diagnostic ▽

▽

△

Class	***DiagnosticServiceInstance*** (abstract)			
	△ RequestControlOfOnBoardDevice, DiagnosticRequestCurrentPowertrainData, DiagnosticRequest EmissionRelatedDTC, DiagnosticRequestEmissionRelatedDTCPermanentStatus, DiagnosticRequestFile Transfer, DiagnosticRequestOnBoardMonitoringTestResults, DiagnosticRequestPowertrainFreezeFrame Data, DiagnosticRequestVehicleInfo, DiagnosticResponseOnEvent, DiagnosticRoutineControl, DiagnosticSecurityAccess, DiagnosticSessionControl			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
access Permission	DiagnosticAccess Permission	0..1	ref	This represents the collection of DiagnosticAccess Permissions that allow for the execution of the referencing DiagnosticServiceInstance..
serviceClass	DiagnosticServiceClass	0..1	ref	This represents the corresponding "class", i.e. this meta-class provides properties that are shared among all instances of applicable sub-classes of DiagnosticService Instance. The subclasses that affected by this pattern implement references to the applicable "class"-role that substantiate this abstract reference. **Stereotypes:** atpAbstract

Table A.77: DiagnosticServiceInstance

Class	**DiagnosticServiceSwMapping**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::ServiceMapping			
Note	This represents the ability to define a mapping of a diagnostic service to a software-component or a basic-software module. If the former is used then this kind of service mapping is applicable for the usage of ClientServerInterfaces. **Tags:**atp.recommendedPackage=DiagnosticServiceMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *DiagnosticSwMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
diagnosticData Element	DiagnosticDataElement	0..1	ref	This represents a DiagnosticDataElement required to execute the respective diagnostic service in the context of the diagnostic service mapping,
mappedBsw Service Dependency	BswService DependencyIdent	0..1	ref	This is supposed to represent a reference to a Bsw ServiceDependency. the latter is not derived from Referrable and therefore this detour needs to be implemented to still let BswServiceDependency become the target of a reference.
mappedFlatSwc Service Dependency	SwcService Dependency	0..1	ref	This represents the ability to refer to an AtomicSw ComponentType that is available without the definition of how it will be embedded into the component hierarchy.
mappedSwc Service DependencyIn Executable	SwcService Dependency	0..1	iref	This represents the ability to point into the component hiearchy of an adaptive AUTOSAR model (under possible consideration of the rootSoftwareComposition) **Tags:**atp.Status=draft
mappedSwc Service DependencyIn System	SwcService Dependency	0..1	iref	This represents the ability to point into the component hiearchy (under possible consideration of the root SoftwareComposition)

▽

△

Class	**DiagnosticServiceSwMapping**			
process	ProcessDesign	0..1	ref	Reference to the representation of a Process that is required because the mapping could be different for different Processes referring to a specific Executable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft
serviceInstance	DiagnosticService Instance	0..1	ref	This represents the service instance that needs to be considered in this diagnostics service mapping.

Table A.78: DiagnosticServiceSwMapping

Class	**DiagnosticServiceValidationInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to process requests for service validation on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.79: DiagnosticServiceValidationInterface

Class	**DiagnosticSession**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dcm			
Note	This meta-class represents the ability to define a diagnostic session. **Tags:**atp.recommendedPackage=DiagnosticSessions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
id	PositiveInteger	1	attr	This is the numerical identifier used to identify the DiagnosticSession in the scope of diagnostic workflow
p2ServerMax	TimeValue	1	attr	This is the session value for P2ServerMax in seconds (per Session Control). The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds.
p2StarServer Max	TimeValue	1	attr	This is the session value for P2*ServerMax in seconds (per Session Control). The AUTOSAR configuration standard is to use SI units, so this parameter is defined as a float value in seconds.

Table A.80: DiagnosticSession

Enumeration	**DiagnosticStatusBitHandlingTestFailedSinceLastClearEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::DiagnosticCommonProps
Note	Aging and displacement has no impact on the "TestFailedSinceLastClear" status bits.
Literal	*Description*
statusBitAgingAnd Displacement	**Tags:**atp.EnumerationLiteralIndex=0
statusBitNormal	**Tags:**atp.EnumerationLiteralIndex=1

Table A.81: DiagnosticStatusBitHandlingTestFailedSinceLastClearEnum

Class	**DiagnosticTroubleCodeGroup**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	The diagnostic trouble code group defines the DTCs belonging together and thereby forming a group. **Tags:**atp.recommendedPackage=DiagnosticTroubleCodes			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*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.82: DiagnosticTroubleCodeGroup

Class	**DiagnosticTroubleCodeProps**			
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode			
Note	This element defines common Dtc properties that can be reused by different non OBD-relevant DTCs. **Tags:**atp.recommendedPackage=DiagnosticTroubleCodePropss			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
aging	DiagnosticAging	0..1	ref	Reference to an aging algorithm in case that an aging/ unlearning of the event is allowed.
agingAllowed	Boolean	0..1	attr	This represents the decision whether aging is allowed for this DiagnosticTroubleCodeProps.
environment CaptureTo Reporting	EnvironmentCaptureTo ReportingEnum	0..1	attr	This attribute determines the point in time, when the data actually is captured.
extendedData Record	DiagnosticExtended DataRecord	*	ref	Defines the links to an extended data class sampler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=extendedDataRecord, variationPoint.short Label vh.latestBindingTime=preCompileTime

△

Class	**DiagnosticTroubleCodeProps**			
freezeFrame	DiagnosticFreezeFrame	*	ref	Define the links to a freeze frame class sampler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=freezeFrame, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
immediateNv DataStorage	Boolean	0..1	attr	Switch to enable immediate storage triggering of an according event memory entry persistently to NVRAM. true: immediate non-volatile storage triggering enabled false: immediate non-volatile storage triggering disabled
legislated FreezeFrame ContentWwh Obd	DiagnosticDataIdentifier Set	0..1	ref	This reference identifies the layout of the WWH-OBD freeze frame. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
maxNumber FreezeFrame Records	PositiveInteger	0..1	attr	This attribute defines the number of according freeze frame records, which can maximal be stored for this event. Therefore all these freeze frame records have the same freeze frame class.
memory Destination	DiagnosticMemory Destination	*	ref	The event destination assigns events to none, one or multiple origins.
priority	PositiveInteger	1	attr	Priority of the event, in view of full event buffer. A lower value means higher priority. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
significance	DiagnosticSignificance Enum	0..1	attr	Significance of the event, which indicates additional information concerning fault classification and resolution.
snapshot RecordContent	DiagnosticDataIdentifier Set	0..1	ref	This represents the freeze frame layout as a set of DIDs. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table A.83: 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***	***Mult.***	***Kind***	***Note***
considerPto Status	Boolean	0..1	attr	This attribute describes the affection of the event by the Dem PTO handling. True: the event is affected by the Dem PTO handling. False: the event is not affected by the Dem PTO handling.
dtcProps	DiagnosticTroubleCode Props	0..1	ref	Defined properties associated with the DemDTC.
eventObd Readiness Group	NameToken	0..1	attr	This attribute specifies the Event OBD Readiness group for PID $01 and PID $41 computation. This attribute is only applicable for emission-related ECUs.

▽

△

Class	**DiagnosticTroubleCodeUds**			
functionalUnit	PositiveInteger	0..1	attr	This attribute specifies a 1-byte value which identifies the corresponding basic vehicle / system function which reports the DTC. This parameter is necessary for the report of severity information.
severity	DiagnosticUdsSeverity Enum	0..1	attr	DTC severity according to ISO 14229-1.
udsDtcValue	PositiveInteger	0..1	attr	Unique Diagnostic Trouble Code value for UDS. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
wwhObdDtc Class	DiagnosticWwhObdDtc ClassEnum	0..1	attr	This attribute is used to identify (if applicable) the corresponding severity class of an WWH-OBD DTC. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table A.84: DiagnosticTroubleCodeUds

Class	**DiagnosticTroubleCodeUdsToClearConditionGroupMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::DiagnosticDesign::DiagnosticClearCondition			
Note	This meta-class provides the ability to map a DiagnosticClearConditionGroup to a collection of Diagnostic TroubleCodeUds. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *DiagnosticCommonElement*, *DiagnosticMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clearCondition Group	DiagnosticClear ConditionGroup	0..1	ref	Thi reference identifies the applicable DiagnosticClear ConditionGroup. **Tags:**atp.Status=draft
troubleCodeUds	DiagnosticTroubleCode Uds	0..1	ref	This reference identifies the DiagnosticTroubleCodeUds that are relevant for the mapping. **Tags:**atp.Status=draft

Table A.85: DiagnosticTroubleCodeUdsToClearConditionGroupMapping

Enumeration	**DiagnosticTypeOfFreezeFrameRecordNumerationEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticTroubleCode
Note	FreezeFrame record numeration type
Literal	***Description***
calculated	Freeze frame records will be numbered consecutive starting by 1 in their chronological order. **Tags:**atp.EnumerationLiteralIndex=0
configured	Freeze frame records will be numbered based on the given configuration in their chronological order. **Tags:**atp.EnumerationLiteralIndex=1

Table A.86: DiagnosticTypeOfFreezeFrameRecordNumerationEnum

Class	**DiagnosticUploadInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::DiagnosticPortInterface			
Note	This meta-class represents the ability to implement a PortInterface to process requests for uploading data using diagnostic channels on the adaptive platform. **Tags:** atp.Status=draft atp.recommendedPackage=DiagnosticPortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DiagnosticPortInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.87: DiagnosticUploadInterface

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***	***Mult.***	***Kind***	***Note***
writeClass	DiagnosticWriteDataBy IdentifierClass	1	ref	This reference substantiates that abstract reference in the role serviceClass for this specific concrete class. Thereby, the reference represents the ability to access shared attributes among all DiagnosticWriteDataBy Identifier in the given context.

Table A.88: DiagnosticWriteDataByIdentifier

Class	**DoIpNetworkConfiguration**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::AdaptiveModule Implementation			
Note	This element collects DoIP properties that are network interface specific. **Tags:** atp.ManifestKind=MachineManifest atp.Status=draft			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
eidUseMac	Boolean	0..1	attr	This attribute defines whther the MAC of the network interface is used as eid. True: MAC is used False: eid needs to be configured manually by DoIpInstantiation.eid.
isActivationLine Dependent	Boolean	1	attr	This attribute defines whether the network interface • is started "on-demand" when an activation line is sensed or • is always available.

▽

△

Class	**DoIpNetworkConfiguration**			
maxInitial Vehicle Announcement Time	TimeValue	1	attr	Upper bound for the time to wait in [s] for sending first vehicle anouncement message after IP address assignment. Represents parameter A_DoIP_Announce_ Wait of ISO 13400-2:2012. The value of this timing shall be determined randomly in the closed interval [0..max InitialVehicleAnnouncementTime].
maxTester Connections	PositiveInteger	1	attr	Maximum amount of tester connections that shall be maintained at one time before alive check is performed.
network Configuration	EthernetNetwork Configuration	0..1	aggr	Network configuration (Protocol, Port, IP Address) for transmission of DoIP messages on a specific VLAN. **Tags:**atp.Status=draft
network InterfaceId	PositiveInteger	1	attr	This attribute defines the identifier for the DoIPInterface.
tcpAliveCheck Response Timeout	TimeValue	0..1	attr	Timeout in [s] for waiting for a response to an Alive Check request before the connection is considered to be disconnected. Represents parameter T_TCP_AliveCheck of ISO 13400-2:2012.
tcpGeneral InactivityTime	TimeValue	0..1	attr	Timeout in [s] for maximum inactivity of a TCP socket connection before the DoIP module will close the according socket connection. Represents parameter T_ TCP_General_Inactivity of ISO 13400-2:2012.
tcpInitial InactivityTime	TimeValue	0..1	attr	Timeout in [s] used for initial inactivity of a connected TCP socket connection directly after socket connection. Represents parameter T_TCP_Initial_Inactivity of ISO 13400-2:2012.
vehicle Announcement Count	PositiveInteger	0..1	attr	Number of vehicle announcement messages on IP address assignment. Represents parameter A_DoIP_ Announce_Num of ISO 13400-2:2012.
vehicle Announcement Interval	TimeValue	0..1	attr	Time to wait in [s] for sending subsequent vehicle anouncement messages. Represents parameter A_Do IP_Announce_Interval of ISO 13400-2:2012.
vehicle Identification SyncStatus	Boolean	1	attr	Defines if the optional VIN/GID synchronization status is used additionally in the vehicle identification/ announcement.

Table A.89: DoIpNetworkConfiguration

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, *AdaptiveModuleInstantiation*, AdaptiveSwcInternal Behavior, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, Autosar OperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModule Dependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *Communication Connector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, Coupling ▽

▽

△

Class	***Identifiable*** (abstract)			
	△ Port, *CouplingPortStructuralElement*, CryptoKeySlot, *CryptoServiceMapping*, DataPrototypeGroup, Data Transformation, DdsRpcServiceDeployment, DependencyOnArtifact, DeterministicClientResourceNeeds, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, Diagnostic FunctionInhibitSource, DiagnosticMasterToSlaveEventMapping, *DiagnosticRoutineSubfunction*, Dlt Argument, DltLogChannel, DltMessage, DoIpInterface, DoIpLogicAddress, E2EProfileConfiguration, EC UMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinition Element*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, End2EndEventProtectionProps, EndToEndProtection, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMap Element, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForget Mapping, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPin Group, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InterfaceMapping, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, Life CycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, McDataInstance, Memory Section, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, Network Endpoint, *NmCluster*, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduTo FrameMapping, PduTriggering, PerInstanceMemory, PersistencyFileProxy, PersistencyKeyValuePair, *PhmActionItem*, PhmActionList, PhmLogicalExpression, PhmRule, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ProcessDesignToMachineDesignMapping, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, RawDataStream MethodDeployment, ResourceConsumption, ResourceGroup, *RestAbstractEndpoint*, RestElementDef, RestResourceDef, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSw CompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, Sec OcJobMapping, SecOcJobRequirement, *SecureComProps*, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, *ServerCallPoint*, *Service EventDeployment*, *ServiceFieldDeployment*, ServiceInstanceToSignalMapping, *ServiceInterfaceElement Mapping*, ServiceInterfaceElementSecureComConfig, ServiceInterfaceMapping, *ServiceMethod Deployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTp Channel, *SpecElementReference*, *StackUsage*, StartupConfig, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcTo ApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, SystemMemory Usage, TcpOptionFilterList, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsJobMapping, Topic1, Tp Address, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationPropsTo ServiceInterfaceElementMapping, TransformationTechnology, Trigger, UcmDescription, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
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
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

▽

△

Class	***Identifiable*** (abstract)			
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:**xml.attribute=true

Table A.90: Identifiable

Class	***ImplementationProps*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as (depending on the concrete case) either a complete replacement or a prefix when generating code artifacts.			
Base	*ARObject*, *Referrable*			
Subclasses	BswSchedulerNamePrefix, ExecutableEntityActivationReason, SectionNamePrefix, SymbolProps, SymbolicNameProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
symbol	CIdentifier	1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table A.91: ImplementationProps

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**	**Mult.**	**Kind**	**Note**
provided Interface	PortInterface	1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table A.92: PPortPrototype

Class	***PortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ClientServerInterface, CompositeInterface, *DataInterface*, *DiagnosticPortInterface*, ModeSwitchInterface, *PersistencyInterface*, *PlatformHealthManagementInterface*, RawDataStreamInterface, RestService Interface, ServiceInterface, *TimeSynchronizationInterface*, TriggerInterface			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName atp.Status=draft

Table A.93: PortInterface

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**	**Mult.**	**Kind**	**Note**
required Interface	PortInterface	1	tref	The interface that this port requires. **Stereotypes:** isOfType

Table A.94: RPortPrototype

Class	ServiceInterface
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface

▽

△

Class	**ServiceInterface**			
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***	***Mult.***	***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 xml.sequenceOffset=30
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 xml.sequenceOffset=40
majorVersion	PositiveInteger	0..1	attr	Major version of the service contract. **Tags:** atp.Status=draft xml.sequenceOffset=10
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 xml.sequenceOffset=50
minorVersion	PositiveInteger	0..1	attr	Minor version of the service contract. **Tags:** atp.Status=draft xml.sequenceOffset=20

Table A.95: ServiceInterface

Class	**SoftwareCluster**
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage
Note	This meta-class represents the ability to define an uploadable software-package, i.e. the SoftwareCluster shall contain all software and configuration for a given purpose. **Tags:** atp.ManifestKind=SoftwareDistribution atp.Status=draft atp.recommendedPackage=SoftwareClusters
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *SoftwareActivationDependency*

▽

△

Class	**SoftwareCluster**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
contained ARElement	ARElement	*	ref	This reference represents the collection of model elements that cannot derive from UploadablePackage Element and that contribute to the completeness of the definition of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=containedARElement atp.Status=draft
containedFibex Element	FibexElement	*	ref	This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster. **Tags:**atp.Status=draft
contained Package Element	UploadablePackage Element	*	ref	This reference identifies model elements that are required to complete the manifest content. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=containedPackageElement atp.Status=draft
contained Process	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster. **Tags:**atp.Status=draft
design	SoftwareClusterDesign	*	ref	This reference represents the identification of all Software ClusterDesigns applicable for the enclosing Software Cluster. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
diagnostic Address	SoftwareCluster DiagnosticAddress	*	aggr	This aggregation represents the collection of diagnostic addresses that apply for the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=diagnosticAddress atp.Status=draft
diagnostic Extract	DiagnosticContribution Set	0..1	ref	This reference represents the definition of the diagnostic extract applicable to the referencing SoftwareCluster **Tags:**atp.Status=draft
license	Documentation	*	ref	This attribute allows for the inclusion of the the full text of a license of the enclosing SoftwareCluster. In many cases open source licenses require the inclusion of the full license text to any software that is released under the respective license. **Tags:**atp.Status=draft
module Instantiation	AdaptiveModule Instantiation	*	ref	This reference identifies AdaptiveModuleInstantiations that need to be included with the SoftwareCluster in order to establish infrastructure required for the installation of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation atp.Status=draft
releaseNotes	Documentation	0..1	ref	This attribute allows for the explanations of changes since the previous version. The list of changes might require the creation of multiple paragraphs of test. **Tags:**atp.Status=draft

▽

△

Class	**SoftwareCluster**			
subSoftware Cluster	SoftwareCluster	*	ref	This reference is used to identify the sub-Software Clusters of an "umbrella" SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=subSoftwareCluster atp.Status=draft
typeApproval	String	0..1	attr	This attribute carries the homologation information that may be specific for a given country.
vendorId	PositiveInteger	1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list.
vendor Signature	CryptoService Certificate	1	ref	This reference identifies the certificate that represents the vendor's signature. **Tags:**atp.Status=draft
version	StrongRevisionLabel String	1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster.

Table A.96: SoftwareCluster

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

Table A.97: SoftwareClusterDiagnosticAddress

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

▽

△

Enumeration	**SoftwareClusterDiagnosticAddressSemanticsEnum**
physicalAddress	This address represents a physical address. **Tags:**atp.EnumerationLiteralIndex=0

Table A.98: SoftwareClusterDiagnosticAddressSemanticsEnum

Class	<<atpVariation>> **SwDataDefProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the Data Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, sw AddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNative TypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:**vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:**xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
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

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:**xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:**xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:**xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:**xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:**xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memory AllocationKeywordPolicy of the referenced SwAddr Method. **Tags:**xml.sequenceOffset=33
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:**xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:**xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:**xml.sequenceOffset=90

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:**xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:**xml.sequenceOffset=230
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:**xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:**xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:**xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:**xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:**xml.sequenceOffset=290
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system.

▽

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
				△ So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:**xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:**xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
swValueBlock SizeMult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:**xml.sequenceOffset=350
valueAxisData Type	ApplicationPrimitive DataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:**xml.sequenceOffset=355

Table A.99: 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***	***Mult.***	***Kind***	***Note***
assignedData	RoleBasedData Assignment	*	aggr	Defines the role of an associated data object of the same component. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcServiceDependency**			
assignedPort	RoleBasedPort Assignment	*	aggr	Defines the role of an associated port of the same component. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedPort, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
representedPort Group	PortGroup	0..1	ref	This reference specifies an association between the ServiceNeeeds and a PortGroup, for example to request a communication mode which applies for communication via these ports. The referred PortGroup shall be local to this atomic SWC, but via the links between the Port Groups, a tool can evaluate this information such that all the ports linked via this port group on the same ECU can be found.
serviceNeeds	ServiceNeeds	1	aggr	The associated ServiceNeeds.

Table A.100: SwcServiceDependency

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to contribute a part of a namespace.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.101: SymbolProps

B History of Constraints and Specification Items

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

B.1 Constraint and Specification Item History of this document according to AUTOSAR Release 17-10

B.1.1 Added Traceables in 17-10

Number	Heading
[SWS_DM_00277]	Cancellation of `Active Protocol` in case of External Service Processing
[SWS_DM_00278]	Cancellation of `Active Protocol` in case of Internal Processing
[SWS_DM_00279]	Cancellation of `Active Protocol` before Response Transmission
[SWS_DM_00280]	Cancellation of `Active Protocol` during Response Transmission
[SWS_DM_00281]	Cancellation of `Active Protocol` in Non-Default Session
[SWS_DM_00282]	Handling of CurrentActiveProtocols
[SWS_DM_00284]	SecurityAccess Service Interface
[SWS_DM_00286]	Configurable environmental condition check execution
[SWS_DM_00287]	Configurable environmental condition check criteria
[SWS_DM_00288]	Configurable environmental condition check evaluates to `TRUE`
[SWS_DM_00289]	Configurable environmental condition check evaluates to `FALSE`
[SWS_DM_00290]	Refusal of second diagnostic request from different diagnostic client without response
[SWS_DM_00291]	UdsMessage class
[SWS_DM_00292]	UdsMessage non public constructors
[SWS_DM_00293]	UdsMessage Address type
[SWS_DM_00294]	meta info map type
[SWS_DM_00295]	meta info map vendor type
[SWS_DM_00296]	TargetAddressType Address type
[SWS_DM_00297]	GetSa method
[SWS_DM_00298]	GetTa method
[SWS_DM_00299]	GetTaType method
[SWS_DM_00300]	GetPayload method readonly
[SWS_DM_00301]	GetPayload method
[SWS_DM_00302]	AddMetaInfo method
[SWS_DM_00303]	UdsMessage Pointer
[SWS_DM_00304]	Const UdsMessage Pointer
[SWS_DM_00305]	Const UdsMessage Pointer vendor type
[SWS_DM_00306]	UdsTransportProtocolMgr class
[SWS_DM_00307]	TransmissionResult type
[SWS_DM_00308]	Global Channel Identifier type
[SWS_DM_00309]	IndicateMessage method
[SWS_DM_00310]	NotifyMessageFailure method
[SWS_DM_00311]	HandleMessage method

▽

△

Number	Heading
[SWS_DM_00312]	TransmitConfirmation method
[SWS_DM_00313]	ChannelReestablished method
[SWS_DM_00314]	HandlerStopped method
[SWS_DM_00315]	UdsTransportProtocolHandler class
[SWS_DM_00316]	Header file
[SWS_DM_00317]	UdsTransportProtocolHandler constructor
[SWS_DM_00318]	UdsTransportProtocolHandler destructor
[SWS_DM_00319]	Initialize method
[SWS_DM_00320]	UdsTransportProtocolHandler UdsTransportProtocolMgr member
[SWS_DM_00321]	constructor member initialization
[SWS_DM_00322]	Start method
[SWS_DM_00323]	Stop method
[SWS_DM_00324]	UdsTransportProtocolHandler UdsTransportProtocolHandlerID member
[SWS_DM_00325]	GetHandlerID method
[SWS_DM_00326]	NotifyReestablishment method
[SWS_DM_00327]	Transmit method
[SWS_DM_00328]	UdsMessage Pointer vendor type
[SWS_DM_00329]	Lifecycle management of an `Uds Transport Protocol` implementation
[SWS_DM_00330]	Construction of an `Uds Transport Protocol` implementation
[SWS_DM_00331]	Initialization of an `Uds Transport Protocol` implementation
[SWS_DM_00332]	Starting of an `Uds Transport Protocol` implementation
[SWS_DM_00333]	Stopping of an `Uds Transport Protocol` implementation
[SWS_DM_00334]	UdsTransportProtocolMgr may be an abstract class
[SWS_DM_00335]	Header file
[SWS_DM_00336]	UdsTransportProtocolHandlerID
[SWS_DM_00337]	ChannelID
[SWS_DM_00338]	ByteVector
[SWS_DM_00339]	ByteVector vendor type
[SWS_DM_00340]	Waiting for Stop confirmation
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00342]	Indication of UDS message reception
[SWS_DM_00343]	Acceptance of UDS message reception
[SWS_DM_00344]	Refusal of UDS message reception
[SWS_DM_00345]	Forwarding of UDS message
[SWS_DM_00346]	Aborting of UDS message
[SWS_DM_00347]	Channel identification in Indication
[SWS_DM_00348]	Transmission of UDS response message

▽

△

Number	Heading
[SWS_DM_00349]	Reuse channel identifier of Indication
[SWS_DM_00350]	Confirmation of UDS message transmission
[SWS_DM_00351]	Confirmation Result
[SWS_DM_00356]	Requesting Notification of a channel reestablishment
[SWS_DM_00357]	Validity/lifetime of a Notification Request
[SWS_DM_00358]	Notification of a channel reestablishment
[SWS_DM_00359]	Persistent Storage of Notification Request
[SWS_DM_00360]	EcuReset positive response processing after reset
[SWS_DM_00361]	EcuReset application error processing
[SWS_DM_00362]	Checking Supported Subfunction for CompareKey
[SWS_DM_00363]	Positive response processing
[SWS_DM_00364]	Negative response processing
[SWS_DM_00365]	Suppression of response
[SWS_DM_00366]	Suppression of response for functional requests
[SWS_DM_00367]	No service processing
[SWS_DM_00368]	Sending busy responses
[SWS_DM_00369]	Max. number of busy responses
[SWS_DM_00370]	Support of UDS service ReadDTCInformation, Subfunction 0x06
[SWS_DM_00371]	Support of UDS service ReadDTCInformation, Subfunction 0x14
[SWS_DM_00372]	Support of UDS service ReadDTCInformation, Subfunction 0x17
[SWS_DM_00373]	Support of UDS service ReadDTCInformation, Subfunction 0x18
[SWS_DM_00374]	Support of UDS service ReadDTCInformation, Subfunction 0x19

Table B.1: Added Traceables in 17-10

B.1.2 Changed Traceables in 17-10

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00004]	Operation cycle persistency
[SWS_DM_00019]	Internal debounce counter incrementation
[SWS_DM_00020]	Internal debounce counter decrementation
[SWS_DM_00023]	Debounce counter jump down behavior
[SWS_DM_00030]	Calculation of the FDC based on the internal debounce timer
[SWS_DM_00042]	Canceling external service processors
[SWS_DM_00043]	Request refusal in case of no resources

▽

△

Number	Heading
[SWS_DM_00044]	Request refusal in case of non-default session active
[SWS_DM_00045]	Ignore ISO same resource access check
[SWS_DM_00046]	Each `Diagnostic Protocol` has own session resources
[SWS_DM_00047]	Each `Diagnostic Protocol` has own security-level resources
[SWS_DM_00048]	Request refusal in case of no resources
[SWS_DM_00049]	Refusal of second diagnostic request from different diagnostic client with BusyRepeatRequest
[SWS_DM_00051]	Cancellation of `Active Protocol` with lower priority
[SWS_DM_00052]	Selection between multiple cancellation candidates
[SWS_DM_00066]	Monitor initialization
[SWS_DM_00072]	Availability of enable condition service interfaces
[SWS_DM_00074]	Unsatisfied enable conditions
[SWS_DM_00088]	ControlDTCSetting influence
[SWS_DM_00089]	Reporting PREPASSED or PREFAILED for events without assigned debouncing algorithm
[SWS_DM_00096]	Validation Steps and Order
[SWS_DM_00098]	UDS message checks
[SWS_DM_00099]	Supported Service SID level checks
[SWS_DM_00100]	Supported Service subfunction level checks
[SWS_DM_00101]	Session Access SID level Permission
[SWS_DM_00102]	Session Access subfunction level Permission
[SWS_DM_00103]	Security Access level Permission
[SWS_DM_00105]	Configurable Manufacturer Permission Check Services
[SWS_DM_00106]	Signature of Manufacturer Permission Check Method
[SWS_DM_00107]	Configurable Supplier Permission Check Services
[SWS_DM_00108]	Signature of Supplier Permission Check Method
[SWS_DM_00111]	Configurable environment condition checks
[SWS_DM_00112]	Condition check definition
[SWS_DM_00136]	Request upload service processing
[SWS_DM_00148]	Persistent storage of event memory entries
[SWS_DM_00153]	Triggering for snapshot record storage
[SWS_DM_00156]	Triggering for extended data record storage and updates
[SWS_DM_00166]	Trigger to process event status
[SWS_DM_00167]	Ignoring reported events for not started operation cycles
[SWS_DM_00169]	Restart of operation cycles
[SWS_DM_00172]	Reaction on Unsupported DataIdentifier
[SWS_DM_00176]	External ReadDataByIdentifier processing
[SWS_DM_00177]	Negative Response processing

▽

△

Number	Heading
[SWS_DM_00179]	Positive Response processing
[SWS_DM_00180]	Provide Protocol Priority Configurability
[SWS_DM_00182]	Identification of a protocol for Priority Assignment
[SWS_DM_00184]	Protocol Match Search
[SWS_DM_00188]	Reaction on Unsupported DataIdentifier
[SWS_DM_00189]	WriteDataByIdentifier processing
[SWS_DM_00192]	Operation cycles are only ended once
[SWS_DM_00202]	Check for Supported RoutineIdentifier and Reaction
[SWS_DM_00203]	Check for Supported Subfunction and Reaction
[SWS_DM_00205]	Providing the `VIN` in DoIP protocol messages
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00249]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00252]	Reaction on Unsupported Subfunction
[SWS_DM_00258]	Cancellation of `Active Protocol` in non-default session
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00269]	Reaction on Unsupported Subfunction
[SWS_DM_00270]	Counting of attempts to change security level
[SWS_DM_00271]	Evaluate the number of failed security level change attempts
[SWS_DM_00272]	Expiration of the delay timer
[SWS_DM_00273]	Notification event upon `snapshot record` updates
[SWS_DM_00274]	Definition of an active `Diagnostic Protocol`

Table B.2: Changed Traceables in 17-10

B.1.3 Deleted Traceables in 17-10

Number	Heading
[SWS_DM_00001]	Availability of operation cycle service interfaces
[SWS_DM_00053]	Cancellation of `Active Protocol`
[SWS_DM_00054]	Generic UDS Service Interface
[SWS_DM_00073]	Checking enable conditions after status reports
[SWS_DM_00075]	Fulfilled enable conditions
[SWS_DM_00076]	Checking storage conditions in case the storage of event-related data is triggered
[SWS_DM_00077]	Checking storage conditions in case the update of event-related data is triggered

▽

△

Number	Heading
[SWS_DM_00081]	Routine Service Interface
[SWS_DM_00093]	Service Validation Interface
[SWS_DM_00094]	Data Services Interface
[SWS_DM_00149]	DTC related data
[SWS_DM_00157]	Snapshot record record data layout
[SWS_DM_00171]	Check for Supported DataIdentifier
[SWS_DM_00187]	Check for Supported DataIdentifier
[SWS_DM_00204]	Reaction on Unsupported Subfunction
[SWS_DM_00251]	Check for Supported Subfunction
[SWS_DM_CONSTR_00275]	Response processing after the actual reset

Table B.3: Deleted Traceables in 17-10

B.1.4 Added Constraints in 17-10

none

B.1.5 Changed Constraints in 17-10

none

B.1.6 Deleted Constraints in 17-10

none

B.2 Constraint and Specification Item History of this document according to AUTOSAR Release 18-03

B.2.1 Added Traceables in 18-03

Number	Heading
[SWS_DM_00001]	SRS Diagnostics
[SWS_DM_00376]	Positive response processing
[SWS_DM_00377]	Enable condition influence on debouncing behavior (reset)

▽

△

Number	Heading
[SWS_DM_00378]	ControlDTCSetting influence (reset)
[SWS_DM_00379]	Handling of storage conditions
[SWS_DM_00380]	Support for S3 timer
[SWS_DM_00381]	Session timeout
[SWS_DM_00382]	Session timeout start
[SWS_DM_00383]	Session timeout stop
[SWS_DM_00384]	IndicationResult type
[SWS_DM_00385]	Acceptance of UDS message reception
[SWS_DM_00386]	Ignoring UDS message reception because DM is busy
[SWS_DM_00387]	Ignoring UDS message reception because DM has no (memory) ressources
[SWS_DM_00388]	Filling provided UdsMessage
[SWS_DM_00389]	Skipping Forwarding of UDS message
[SWS_DM_00390]	Dispatching physical Request
[SWS_DM_00391]	Dispatching functional Request
[SWS_DM_00392]	Properties of returned UdsMessage
[SWS_DM_00393]	Retrieving data for `internal DiagnosticDataElement`s
[SWS_DM_00397]	Retrieving data for `external DiagnosticDataElement`s
[SWS_DM_00401]	Reading Diagnostic Data Identifier on Data Element level
[SWS_DM_00402]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00403]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00404]	Default Service Interface for reading `DiagnosticDataIdentifier`
[SWS_DM_00405]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00406]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00407]	Default Service Interface for writing `DiagnosticDataIdentifier`
[SWS_DM_00408]	Retrieving data for requested DataIdentifier
[SWS_DM_00409]	Check supported DataIdentifier
[SWS_DM_00410]	Check session permission
[SWS_DM_00411]	Check security level permission
[SWS_DM_00412]	Check requested number of DataIdentifiers
[SWS_DM_00413]	Check supported DataIdentifier in active session
[SWS_DM_00414]	Check supported DataIdentifier on active security level
[SWS_DM_00415]	Check supported DataIdentifier
[SWS_DM_00416]	Check supported DataIdentifier in active session
[SWS_DM_00417]	Check supported DataIdentifier on active security level
[SWS_DM_00418]	Writing data for requested DataIdentifier
[SWS_DM_00419]	Reaction on ApplicationError
[SWS_DM_00420]	Instantiation of Diagnostic Server

▽

△

Number	Heading
[SWS_DM_00434]	Providing the PowerMode in DoIP protocol messages
[SWS_DM_CONSTR_00394]	Internal DiagnosticDataElements are read-only
[SWS_DM_CONSTR_00395]	Restriction on DEM-exclusive DiagnosticDataElements
[SWS_DM_CONSTR_00396]	Restriction on DCM-exclusive DiagnosticDataElements

Table B.4: Added Traceables in 18-03

B.2.2 Changed Traceables in 18-03

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00005]	DoIP Support
[SWS_DM_00007]	Uniqueness of diagnostic events
[SWS_DM_00008]	Diagnostic event processing interface
[SWS_DM_00012]	DoIP configurable source address identification
[SWS_DM_00013]	Events without debouncing
[SWS_DM_00014]	Use of counter-based debouncing for events
[SWS_DM_00015]	Use of timer based debouncing for events
[SWS_DM_00017]	Calculation of the FDC based on the internal debounce counter
[SWS_DM_00018]	Internal debounce counter init and storage
[SWS_DM_00019]	Internal debounce counter incrementation
[SWS_DM_00020]	Internal debounce counter decrementation
[SWS_DM_00021]	Direct failed qualification of counter-based events
[SWS_DM_00022]	Debounce counter jump up behavior
[SWS_DM_00023]	Debounce counter jump down behavior
[SWS_DM_00024]	Qualified failed event using counter-based debouncing
[SWS_DM_00025]	Qualified passed event using counter-based debouncing
[SWS_DM_00026]	Application resetting the debounce counter
[SWS_DM_00028]	Debounce counter persistency
[SWS_DM_00029]	Direct passed qualification of counter-based events
[SWS_DM_00030]	Calculation of the FDC based on the internal debounce timer
[SWS_DM_00031]	Starting time-based event debouncing for failed
[SWS_DM_00032]	Restrictions on restarting a running event debounce timer for failed
[SWS_DM_00033]	Debounce timer behavior upon reported failed

▽

△

Number	Heading
[SWS_DM_00034]	Starting time-based event debouncing for passed
[SWS_DM_00035]	Restrictions on restarting a running event debounce timer for passed
[SWS_DM_00036]	Debounce timer behavior upon reported passed
[SWS_DM_00037]	Debounce time freeze request
[SWS_DM_00038]	Continuing a frozen debounce timer
[SWS_DM_00039]	Resetting the debounce counter upon starting or restarting an operation cycle
[SWS_DM_00040]	Definition of debounce counter reset
[SWS_DM_00041]	Behavior according to ISO Multiple client handling flow
[SWS_DM_00042]	Cancelling external service processors
[SWS_DM_00043]	Request refusal in case of no resources
[SWS_DM_00044]	Request refusal in case of non-default session active
[SWS_DM_00045]	Ignore ISO same resource access check
[SWS_DM_00046]	Each `Diagnostic Protocol` has own session resources
[SWS_DM_00047]	Each `Diagnostic Protocol` has own security-level resources
[SWS_DM_00048]	Request refusal in case of no resources
[SWS_DM_00049]	Refusal of second diagnostic request from different diagnostic client with BusyRepeatRequest
[SWS_DM_00052]	Selection between multiple cancellation candidates
[SWS_DM_00055]	Supported event memories
[SWS_DM_00057]	Availability of a user-defined event memory
[SWS_DM_00058]	DTC interpretation format
[SWS_DM_00060]	Set of supported DTCs
[SWS_DM_00061]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCByStatusMask
[SWS_DM_00062]	Mapping between ISO 14229-1[1] and Autosar Diagnostic Extract Template [2] of the DTCFormatIdentifier
[SWS_DM_00063]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord
[SWS_DM_00064]	Definition of DTC groups
[SWS_DM_00065]	Always supported availability of the group of all DTCs
[SWS_DM_00069]	Monitor initialization for enable condition reenabling reason
[SWS_DM_00070]	Monitor initialization for `DTC` setting re-enabling reason
[SWS_DM_00071]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00074]	Handling of enable conditions
[SWS_DM_00085]	Internal debounce counter init
[SWS_DM_00086]	Resetting the debounce counter after clearing DTC
[SWS_DM_00087]	Enable condition influence on debouncing behavior (freeze)
[SWS_DM_00088]	ControlDTCSetting influence (freeze)

▽

△

Number	Heading
[SWS_DM_00089]	Reporting PREPASSED or PREFAILED for events without assigned debouncing algorithm
[SWS_DM_00090]	Support of UDS service ClearDiagnosticInformation
[SWS_DM_00091]	Evaluation of ClearDiagnosticInformation parameters
[SWS_DM_00092]	Parameter range check for groupOfDTC request parameter
[SWS_DM_00096]	Validation Steps and Order
[SWS_DM_00097]	Abort on failed verification step
[SWS_DM_00111]	Configurable environment condition checks
[SWS_DM_00112]	Condition check definition
[SWS_DM_00113]	Positive response for UDS service 0x14
[SWS_DM_00114]	Limitation to one simultaneous DTC clear operation
[SWS_DM_00115]	Memory error handling while clearing DTCs
[SWS_DM_00116]	Clearing a DTC group
[SWS_DM_00117]	Clearing a DTC
[SWS_DM_00118]	Event specific configuration to allow clearing of a DTC
[SWS_DM_00119]	Init value for events with clear allowed information
[SWS_DM_00120]	Description of application interface to control the clear event behavior
[SWS_DM_00121]	Forbidden clearing of snapshot records and extended data records
[SWS_DM_00122]	UDS response behavior on not allowed clear operations
[SWS_DM_00123]	Block status byte clearing during a clear DTC operation
[SWS_DM_00124]	Limited status byte clearing during a clear DTC operation
[SWS_DM_00125]	Linking between event clear allowed and clearing a DTC
[SWS_DM_00128]	Realisation of UDS service 0x34 RequestDownload
[SWS_DM_00129]	Supported addressAndLengthFormatIdentifier
[SWS_DM_00130]	Not supported addressAndLengthFormatIdentifier
[SWS_DM_00136]	Request upload service processing
[SWS_DM_00138]	Transfer data service processing
[SWS_DM_00139]	Transfer data service validation
[SWS_DM_00142]	Transfer data service processing
[SWS_DM_00143]	Transfer data service validation
[SWS_DM_00144]	Parallel clearing DTCs in different DiagnosticMemoryDestination
[SWS_DM_00145]	Allow only one simultaneous clear DTC operation for one DiagnosticMemoryDestination
[SWS_DM_00146]	Unlock clear DTC operation for one DiagnosticMemoryDestination
[SWS_DM_00147]	Behavior while trying to clear DTCs on a locked DiagnosticMemoryDestination
[SWS_DM_00148]	Persistent storage of event memory entries
[SWS_DM_00151]	Snapshot record numeration

▽

△

Number	Heading
[SWS_DM_00152]	Number of snapshot records for a DTC
[SWS_DM_00153]	Triggering for snapshot record storage
[SWS_DM_00154]	Number of extended data for a DTC
[SWS_DM_00155]	Extended data record numeration
[SWS_DM_00156]	Triggering for extended data record storage and updates
[SWS_DM_00159]	Allow only to clear `GroupOfAllDTCs`
[SWS_DM_00160]	Allow to clear single `DTCs`
[SWS_DM_00161]	Negative response on not supported GroupOfDTC parameter
[SWS_DM_00162]	Point in time for positive response for ClearDTC
[SWS_DM_00166]	Trigger to process event status
[SWS_DM_00167]	Ignoring reported events for not started operation cycles
[SWS_DM_00168]	Availability of `DiagnosticMonitor` service interfaces
[SWS_DM_00177]	Reaction on ApplicationError
[SWS_DM_00180]	Provide Protocol Priority Configurability
[SWS_DM_00182]	Identification of a protocol for Priority Assignment
[SWS_DM_00183]	Wildcards per attribute
[SWS_DM_00184]	Protocol Match Search
[SWS_DM_00194]	Definition of the user-defined fault memory number for ClearDiagnosticInformation
[SWS_DM_00202]	Check for Supported RoutineIdentifier and Reaction
[SWS_DM_00203]	Check for Supported Subfunction and Reaction
[SWS_DM_00205]	Providing the `VIN` in DoIP protocol messages
[SWS_DM_00213]	DTC status processing
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00217]	DTC status bit transitions triggered by ClearDiagnosticInformation UDS service
[SWS_DM_00218]	Confirmation
[SWS_DM_00219]	Observability of the status byte
[SWS_DM_00220]	Notification about the changes of the status byte
[SWS_DM_00223]	Handling of 'warningIndicatorRequested' bit
[SWS_DM_00227]	Check for supported sessions
[SWS_DM_00229]	Support of UDS service ControlDTCSetting
[SWS_DM_00230]	Check for supported subfunctions
[SWS_DM_00231]	Invalid value for optional request parameter
[SWS_DM_00232]	Support of Subfunction 0x01 (ON)
[SWS_DM_00233]	Support of Subfunction 0x02 (OFF)

▽

△

Number	Heading
[SWS_DM_00236]	Realization of UDS service 0x27 SecurityAccess
[SWS_DM_00237]	Aging
[SWS_DM_00238]	Aging and healing
[SWS_DM_00239]	Aging counter
[SWS_DM_00240]	Processing the aging counter
[SWS_DM_00241]	Aging cycle and threshold
[SWS_DM_00242]	Reoccurrence after aging
[SWS_DM_00243]	Aging-related UDS status byte processing
[SWS_DM_00244]	Support of UDS service ReadDTCInformation, Subfunction 0x01
[SWS_DM_00245]	Support of UDS service ReadDTCInformation, Subfunction 0x02
[SWS_DM_00246]	Support of UDS service ReadDTCInformation, Subfunction 0x04
[SWS_DM_00247]	Support of UDS service ReadDTCInformation, Subfunction 0x07
[SWS_DM_00248]	Notification about session change
[SWS_DM_00249]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00250]	Notification about security-level change
[SWS_DM_00258]	Cancellation of `Active Protocol` in non-default session
[SWS_DM_00259]	Completion of already `Active Protocol`s in default session
[SWS_DM_00260]	instances of interface ClearDTC
[SWS_DM_00261]	Usage of ClearDTC Interface
[SWS_DM_00262]	Common semantic behavior for ClearDTC triggered via diagnostics or application
[SWS_DM_00265]	ClearDTC called while another clear operation is in progress
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00270]	Counting of attempts to change security level
[SWS_DM_00271]	Evaluate the number of failed security level change attempts
[SWS_DM_00272]	Expiration of the delay timer
[SWS_DM_00273]	Notification event upon `snapshot record` updates
[SWS_DM_00277]	Cancellation of `Active Protocol` in case of External Service Processing
[SWS_DM_00278]	Cancellation of `Active Protocol` in case of Internal Processing
[SWS_DM_00279]	Cancellation of `Active Protocol` before Response Transmission
[SWS_DM_00280]	Cancellation of `Active Protocol` at Response Transmission
[SWS_DM_00281]	Cancellation of active DiagnosticConversation in Non-Default Session
[SWS_DM_00282]	Handling of non-/active diagnostic conversations
[SWS_DM_00286]	Configurable environmental condition check execution
[SWS_DM_00290]	Refusal of second diagnostic request from different diagnostic client without response
[SWS_DM_00309]	IndicateMessage method

▽

△

Number	Heading
[SWS_DM_00316]	Header file
[SWS_DM_00329]	Lifecycle management of an `Uds Transport Protocol` implementation
[SWS_DM_00330]	Construction of an `Uds Transport Protocol` implementation
[SWS_DM_00331]	Initialization of an `Uds Transport Protocol` implementation
[SWS_DM_00332]	Starting of an `Uds Transport Protocol` implementation
[SWS_DM_00333]	Stopping of an `Uds Transport Protocol` implementation
[SWS_DM_00335]	Header file
[SWS_DM_00340]	Waiting for Stop confirmation
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00342]	Indication of UDS message reception
[SWS_DM_00345]	Forwarding of UDS message
[SWS_DM_00346]	Aborting of UDS message
[SWS_DM_00347]	Channel identification in Indication
[SWS_DM_00348]	Transmission of UDS response message
[SWS_DM_00349]	Reuse channel identifier of Indication
[SWS_DM_00350]	Confirmation of UDS message transmission
[SWS_DM_00351]	Confirmation Result
[SWS_DM_00356]	Requesting Notification of a channel reestablishment
[SWS_DM_00357]	Validity/lifetime of a Notification Request
[SWS_DM_00358]	Notification of a channel reestablishment
[SWS_DM_00359]	Persistent Storage of Notification Request
[SWS_DM_00362]	Checking Supported Subfunction for CompareKey
[SWS_DM_00363]	Unsupported Subfunction
[SWS_DM_00366]	Suppression of response for functional requests
[SWS_DM_00369]	Max. number of busy responses
[SWS_DM_00370]	Support of UDS service ReadDTCInformation, Subfunction 0x06
[SWS_DM_00371]	Support of UDS service ReadDTCInformation, Subfunction 0x14
[SWS_DM_00372]	Support of UDS service ReadDTCInformation, Subfunction 0x17
[SWS_DM_00373]	Support of UDS service ReadDTCInformation, Subfunction 0x18
[SWS_DM_00374]	Support of UDS service ReadDTCInformation, Subfunction 0x19
[SWS_DM_CONSTR_00059]	Restriction on supported DTC format
[SWS_DM_CONSTR_00082]	Restriction on the configuration of the DTC group GroupOfAllDTCs
[SWS_DM_CONSTR_00084]	Each DTC shall be assigned to an event memory destination
[SWS_DM_CONSTR_00168]	Required operation cycles for diagnostic events

▽

△

Number	Heading
[SWS_DM_CONSTR_00206]	Supported format for data identifier for VINDataIdentifier
[SWS_DM_CONSTR_00207]	Required VINDataIdentifier

Table B.5: Changed Traceables in 18-03

B.2.3 Deleted Traceables in 18-03

Number	Heading
[SWS_DM_00072]	Availability of enable condition service interfaces
[SWS_DM_00078]	Unsatisfied storage conditions
[SWS_DM_00079]	Fulfilled storage conditions
[SWS_DM_00172]	Reaction on Unsupported DataIdentifier
[SWS_DM_00173]	Classification as Internally implemented DID
[SWS_DM_00174]	Internally implemented DID ActiveDiagnosticSessionDataIdentifier
[SWS_DM_00175]	Classification as Externally implemented DID
[SWS_DM_00176]	External ReadDataByIdentifier processing
[SWS_DM_00178]	Check requested number of DataIdentifiers
[SWS_DM_00179]	Positive Response processing
[SWS_DM_00188]	Reaction on Unsupported DataIdentifier
[SWS_DM_00189]	WriteDataByIdentifier processing
[SWS_DM_00190]	Negative Response processing
[SWS_DM_00191]	Positive Response processing
[SWS_DM_00264]	ClearDTC call on invalid DTCOrigin
[SWS_DM_00292]	UdsMessage non public constructors
[SWS_DM_00343]	Acceptance of UDS message reception
[SWS_DM_00344]	Refusal of UDS message reception

Table B.6: Deleted Traceables in 18-03

B.2.4 Added Constraints in 18-03

none

B.2.5 Changed Constraints in 18-03

none

B.2.6 Deleted Constraints in 18-03

none

B.3 Constraint and Specification Item History of this document according to AUTOSAR Release 18-10

B.3.1 Added Traceables in 18-10

Number	Heading
[SWS_DM_00421]	Identification of a Diagnostic Client
[SWS_DM_00422]	Instantiation of Diagnostic Conversation Service Interface
[SWS_DM_00423]	Assignment of Diagnostic Conversation to Service Instances
[SWS_DM_00424]	Reset Service Instance fields on end of Diagnostic Conversation
[SWS_DM_00425]	Procedure to assign UDS requests to Diagnostic Conversations
[SWS_DM_00426]	Assigning a UDS request to an existing Diagnostic Conversation
[SWS_DM_00427]	Priority of a Diagnostic Conversation
[SWS_DM_00428]	Treatment of priority values
[SWS_DM_00429]	Prioritization in case of Pseudo Parallel Mode and active non-default session
[SWS_DM_00430]	Prioritization against all Diagnostic Conversations
[SWS_DM_00431]	Replacement of Diagnostic Conversations
[SWS_DM_00432]	Initial values for Diagnostic Conversation
[SWS_DM_00433]	Refusal of diagnostic request due to busy Diagnostic Conversation
[SWS_DM_00435]	Default session change trigger from AAs
[SWS_DM_00436]	Providing the GID in DoIP protocol messages
[SWS_DM_00437]	Check supported RoutineIdentifier on active security level
[SWS_DM_00438]	Check Support of UDS service RequestUpload (0x35) in active session
[SWS_DM_00439]	Check Support of UDS service RequestUpload (0x35) on active security level
[SWS_DM_00440]	Check Support of UDS service TransferData (0x36) in active session
[SWS_DM_00441]	Check Support of UDS service TransferData (0x36) on active security level
[SWS_DM_00442]	Check Support of UDS service RequestTransferExit (0x37) in active session
[SWS_DM_00443]	Check Support of UDS service RequestTransferExit (0x37) on active security level
[SWS_DM_00444]	Check Support of UDS service ControlDTCSetting (0x85) in active session
[SWS_DM_00445]	Check Support of UDS service ControlDTCSetting (0x85) on active security level
[SWS_DM_00446]	Check Support of UDS service RequestDownload (0x34) in active session
[SWS_DM_00447]	Check Support of UDS service RequestDownload (0x34) on active security level

▽

△

Number	Heading
[SWS_DM_00448]	Check supported RoutineIdentifier in active session
[SWS_DM_00449]	Supported DoIP message types
[SWS_DM_00451]	
[SWS_DM_00452]	
[SWS_DM_00475]	DoIP Version
[SWS_DM_00476]	User Controlled Warning IndicatorRequest-bit
[SWS_DM_00477]	Not Storing of 'warningIndicatorRequested' bit
[SWS_DM_00478]	Persistent Storage of failed attempts to change security level
[SWS_DM_00479]	Blocking Timer for security access on Restart or Power down - power up cycle
[SWS_DM_00480]	Security Access Blocking Timer
[SWS_DM_00481]	Handling of DiagnosticClearConditions
[SWS_DM_00482]	Cancellation of a Diagnostic Conversation
[SWS_DM_00483]	Cancellation trigger from AAs
[SWS_DM_00484]	Updating DiagnosticConversation Service Instance fields
[SWS_DM_00485]	Reinitialization of Service Instance on Cancellation of a Diagnostic Conversation
[SWS_DM_00487]	Ignoring UDS message reception because of unknown target address
[SWS_DM_00491]	Realisation of UDS service 0x86 ResponseOnEvent
[SWS_DM_00492]	Client Server communication
[SWS_DM_00493]	Reestablishing of Client Server communication
[SWS_DM_00494]	Supported sub functions of ResponseOnEvent service
[SWS_DM_00495]	Start initialisation of ResponseOnEvent
[SWS_DM_00496]	Stop initialisation of ResponseOnEvent
[SWS_DM_00497]	Clear initialisation of ResponseOnEvent
[SWS_DM_00498]	Exclusive ResponseOnEvent ressources
[SWS_DM_00499]	Replacement of a not started ResponseOnEvent initialisation
[SWS_DM_00500]	Replacement of a started ResponseOnEvent initialisation
[SWS_DM_00501]	Behavior while trying ResponseOnEvent activation while ResponseOnEvent is not initialised
[SWS_DM_00503]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00504]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00505]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00506]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00507]	Length check on UDS Service 0x27 request with Subfunction for RequestSeed
[SWS_DM_00508]	Reading DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00509]	Writing DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00651]	NumberOfStoredEntries

▽

△

Number	Heading
[SWS_DM_09010]	
[SWS_DM_09012]	
[SWS_DM_09015]	
[SWS_DM_09016]	
[SWS_DM_09017]	
[SWS_DM_09021]	
[SWS_DM_09028]	
[SWS_DM_CONSTR_00208]	Delay time value for sharedTimer
[SWS_DM_NA]	

Table B.7: Added Traceables in 18-10

B.3.2 Changed Traceables in 18-10

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00004]	Operation cycle persistency
[SWS_DM_00005]	DoIP Support
[SWS_DM_00008]	Diagnostic event processing interface
[SWS_DM_00011]	Selectability of parallelism mode
[SWS_DM_00014]	Use of counter-based debouncing for events
[SWS_DM_00015]	Use of timer based debouncing for events
[SWS_DM_00016]	Configurable number of supported parallel Diagnostic Conversations
[SWS_DM_00020]	Internal debounce counter decrementation
[SWS_DM_00026]	Application resetting the debounce counter
[SWS_DM_00031]	Starting time-based event debouncing for failed
[SWS_DM_00034]	Starting time-based event debouncing for passed
[SWS_DM_00037]	Debounce time freeze request
[SWS_DM_00042]	Canceling external service processors
[SWS_DM_00046]	Each `Diagnostic Conversation` has its own session resources
[SWS_DM_00047]	Each `Diagnostic Conversation` has its own security-level resources
[SWS_DM_00049]	Refusal of diagnostic request due to prioritization with BusyRepeatRequest
[SWS_DM_00061]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCByStatusMask
[SWS_DM_00062]	Mapping between ISO 14229-1[1] and Autosar Diagnostic Extract Template [2] of the DTCFormatIdentifier

▽

△

Number	Heading
[SWS_DM_00063]	Providing rule for DTCFormatIdentifier in positive response ReadDTCInformation.reportNumberOfDTCBySeverityMaskRecord
[SWS_DM_00067]	Monitor initialization for clearing reason
[SWS_DM_00068]	Monitor initialization for operation cycle restart reason
[SWS_DM_00069]	Monitor initialization for enable condition re-enabling reason
[SWS_DM_00070]	Monitor initialization for DTC setting re-enabling reason
[SWS_DM_00071]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00074]	Handling of enable conditions
[SWS_DM_00089]	Reporting kPrepassed or kPrefailed for events without an assigned debouncing algorithm
[SWS_DM_00090]	Support of UDS service ClearDiagnosticInformation
[SWS_DM_00091]	Evaluation of ClearDiagnosticInformation parameters
[SWS_DM_00092]	Parameter range check for groupOfDTC request parameter
[SWS_DM_00096]	Validation Steps and Order
[SWS_DM_00098]	UDS message checks
[SWS_DM_00099]	Supported Service SID level checks
[SWS_DM_00100]	Supported Service subfunction level checks
[SWS_DM_00101]	Session Access SID level Permission
[SWS_DM_00102]	Session Access subfunction level Permission
[SWS_DM_00103]	Security Access level Permission
[SWS_DM_00104]	Supported UDS Services
[SWS_DM_00106]	Signature of Manufacturer Permission Check Method
[SWS_DM_00108]	Signature of Supplier Permission Check Method
[SWS_DM_00111]	Configurable environment condition checks
[SWS_DM_00112]	Condition check definition
[SWS_DM_00113]	Positive response for UDS service 0x14
[SWS_DM_00114]	Limitation to one simultaneous DTC clear operation
[SWS_DM_00115]	Memory error handling while clearing DTCs
[SWS_DM_00117]	Clearing a DTC
[SWS_DM_00121]	Forbidden clearing of snapshot records and extended data records
[SWS_DM_00122]	UDS response behavior on not allowed clear operations
[SWS_DM_00123]	Block status byte clearing during a clear DTC operation
[SWS_DM_00124]	Limited status byte clearing during a clear DTC operation
[SWS_DM_00126]	Realisation of UDS service 0x3E TesterPresent
[SWS_DM_00127]	Availability of diagnostic service processors
[SWS_DM_00128]	Realization of UDS service RequestDownload (0x34)
[SWS_DM_00129]	Supported addressAndLengthFormatIdentifier

▽

△

Number	Heading
[SWS_DM_00130]	Not supported addressAndLengthFormatIdentifier
[SWS_DM_00131]	UDS service RequestDownload (0x34) processing
[SWS_DM_00134]	Realization of UDS service RequestUpload (0x35)
[SWS_DM_00136]	UDS service RequestUpload (0x35) processing
[SWS_DM_00137]	Realization of UDS service TransferData (0x36)
[SWS_DM_00138]	UDS service TransferData (0x36) processing
[SWS_DM_00139]	UDS service TransferData (0x36) validation
[SWS_DM_00140]	Realisation of UDS service 0x28 CommunicationControl
[SWS_DM_00141]	Realization of UDS service RequestTransferExit (0x37)
[SWS_DM_00142]	UDS service RequestTransferExit (0x37) processing
[SWS_DM_00143]	UDS service RequestTransferExit (0x37) validation
[SWS_DM_00153]	Triggering for snapshot record storage
[SWS_DM_00156]	Triggering for extended data record storage and updates
[SWS_DM_00159]	Allow only to clear `GroupOfAllDTCs`
[SWS_DM_00160]	Allow to clear single `DTCs`
[SWS_DM_00162]	Point in time for positive response for ClearDTC
[SWS_DM_00163]	Definition of a failed clear operation with event clear allowed and event combination
[SWS_DM_00164]	Definition of a failed clear operation with event clear allowed and clearing a group of DTCs
[SWS_DM_00167]	Ignoring reported events for not started operation cycles
[SWS_DM_00168]	Availability of `DiagnosticMonitor` service interfaces
[SWS_DM_00169]	Restart of operation cycles
[SWS_DM_00170]	Realisation of UDS service ReadDataByIdentifier (0x22)
[SWS_DM_00177]	Reaction on ApplicationError
[SWS_DM_00186]	Realisation of UDS service WriteDataByIdentifier (0x2E)
[SWS_DM_00192]	Operation cycles are only ended once
[SWS_DM_00193]	Support of a user-defined fault memory clear request
[SWS_DM_00194]	Definition of the user-defined fault memory number for ClearDiagnosticInformation
[SWS_DM_00195]	Clearing a user-defined memory
[SWS_DM_00197]	Communication control service processing
[SWS_DM_00198]	Negative Response processing
[SWS_DM_00199]	Positive Response processing
[SWS_DM_00201]	Realization of UDS service RoutineControl (0x31)
[SWS_DM_00202]	Check for Supported RoutineIdentifier and Reaction
[SWS_DM_00203]	Check for Supported Subfunction and Reaction
[SWS_DM_00205]	Providing the `VIN` in DoIP protocol messages

▽

△

Number	Heading
[SWS_DM_00208]	Validation of the requested user-defined memory number
[SWS_DM_00210]	UDS Service RoutineControl (0x31) startRoutine processing
[SWS_DM_00211]	UDS Service RoutineControl (0x31) requestRoutineResults processing
[SWS_DM_00212]	UDS Service RoutineControl (0x31) stopRoutine processing
[SWS_DM_00213]	DTC status processing
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00216]	DTC status bit transitions triggered by operation cycle changes
[SWS_DM_00217]	DTC status bit transitions triggered by ClearDiagnosticInformation UDS service
[SWS_DM_00218]	Confirmation
[SWS_DM_00219]	Observability of the status byte
[SWS_DM_00220]	Notification about DTC status changes
[SWS_DM_00222]	Observability of indicator status
[SWS_DM_00226]	Support of UDS service DiagnosticSessionControl
[SWS_DM_00227]	Check for supported sessions
[SWS_DM_00228]	Switch to requested Diagnostic Session
[SWS_DM_00229]	Support of UDS service ControlDTCSetting (0x85)
[SWS_DM_00230]	Check for supported subfunctions
[SWS_DM_00231]	Invalid value for optional request parameter
[SWS_DM_00232]	Support of Subfunction 0x01 (ON)
[SWS_DM_00233]	Support of Subfunction 0x02 (OFF)
[SWS_DM_00234]	Support of UDS service ECUReset
[SWS_DM_00235]	ECUReset service processing
[SWS_DM_00236]	Realization of UDS service 0x27 SecurityAccess
[SWS_DM_00237]	Aging
[SWS_DM_00240]	Processing the aging counter
[SWS_DM_00241]	Aging cycle and threshold
[SWS_DM_00242]	Re-occurrence after aging
[SWS_DM_00243]	Aging-related UDS DTC status byte processing
[SWS_DM_00244]	Support of UDS service ReadDTCInformation, Subfunction 0x01
[SWS_DM_00245]	Support of UDS service ReadDTCInformation, Subfunction 0x02
[SWS_DM_00246]	Support of UDS service ReadDTCInformation, Subfunction 0x04
[SWS_DM_00247]	Support of UDS service ReadDTCInformation, Subfunction 0x07
[SWS_DM_00248]	Notification about session change
[SWS_DM_00249]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00250]	Notification about security-level change

▽

△

Number	Heading
[SWS_DM_00252]	Reaction on Unsupported Subfunction
[SWS_DM_00260]	instances of interface ClearDTC
[SWS_DM_00261]	Usage of ClearDTC Interface
[SWS_DM_00263]	ClearDTC call on invalid DTC or DTCgroup
[SWS_DM_00265]	ClearDTC called while another clear operation is in progress
[SWS_DM_00266]	ClearDTC processing in case of memory errors
[SWS_DM_00267]	Possible failure of ClearDTC
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00269]	Reaction on Unsupported Subfunction
[SWS_DM_00270]	Counting of attempts to change security level
[SWS_DM_00271]	Evaluate the number of failed security level change attempts
[SWS_DM_00273]	Notification event upon `snapshot record` updates
[SWS_DM_00277]	Cancellation of a Diagnostic Conversation in case of External Service Processing
[SWS_DM_00278]	Cancellation of a Diagnostic Conversation in case of Internal Processing
[SWS_DM_00279]	Cancellation of a Diagnostic Conversation before Response Transmission
[SWS_DM_00280]	Cancellation of a Diagnostic Conversation at Response Transmission
[SWS_DM_00286]	Configurable environmental condition check execution
[SWS_DM_00288]	Configurable environmental condition check evaluates to `TRUE`
[SWS_DM_00289]	Configurable environmental condition check evaluates to `FALSE`
[SWS_DM_00290]	Refusal of diagnostic request due to prioritization without response
[SWS_DM_00291]	
[SWS_DM_00293]	
[SWS_DM_00294]	
[SWS_DM_00296]	
[SWS_DM_00297]	
[SWS_DM_00298]	
[SWS_DM_00299]	
[SWS_DM_00300]	
[SWS_DM_00301]	
[SWS_DM_00302]	
[SWS_DM_00303]	
[SWS_DM_00304]	
[SWS_DM_00306]	
[SWS_DM_00307]	
[SWS_DM_00309]	
[SWS_DM_00310]	

▽

△

Number	Heading
[SWS_DM_00311]	
[SWS_DM_00312]	
[SWS_DM_00313]	
[SWS_DM_00314]	
[SWS_DM_00315]	
[SWS_DM_00319]	
[SWS_DM_00322]	
[SWS_DM_00323]	
[SWS_DM_00325]	
[SWS_DM_00326]	
[SWS_DM_00327]	
[SWS_DM_00329]	Lifecycle management of an `Uds Transport Protocol` implementation
[SWS_DM_00336]	
[SWS_DM_00337]	
[SWS_DM_00338]	
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00360]	EcuReset positive response processing after reset
[SWS_DM_00361]	EcuReset application error processing
[SWS_DM_00362]	Checking Supported Subfunction for CompareKey
[SWS_DM_00364]	Negative response processing
[SWS_DM_00365]	Suppression of positive response in accordance to ISO 14229-1[1]
[SWS_DM_00366]	Suppression of negative response for functional requests in accordance to ISO 14229-1[1]
[SWS_DM_00367]	No service processing
[SWS_DM_00368]	Sending busy responses
[SWS_DM_00369]	Maximum number of busy responses
[SWS_DM_00370]	Support of UDS service ReadDTCInformation, Subfunction 0x06
[SWS_DM_00371]	Support of UDS service ReadDTCInformation, Subfunction 0x14
[SWS_DM_00372]	Support of UDS service ReadDTCInformation, Subfunction 0x17
[SWS_DM_00373]	Support of UDS service ReadDTCInformation, Subfunction 0x18
[SWS_DM_00374]	Support of UDS service ReadDTCInformation, Subfunction 0x19
[SWS_DM_00376]	Positive response processing
[SWS_DM_00379]	Handling of storage conditions
[SWS_DM_00380]	Support for S3 timer
[SWS_DM_00381]	Session timeout
[SWS_DM_00384]	
[SWS_DM_00385]	Acceptance of UDS message reception

▽

△

Number	Heading
[SWS_DM_00386]	Ignoring UDS message reception because DM is busy
[SWS_DM_00393]	Retrieving data for `internal` `DiagnosticDataElement`s
[SWS_DM_00397]	Retrieving data for `external` `DiagnosticDataElement`s
[SWS_DM_00401]	Reading Diagnostic Data Identifier on Data Element level
[SWS_DM_00404]	Default Service Interface for reading `DiagnosticDataIdentifier`
[SWS_DM_00407]	Default Service Interface for writing `DiagnosticDataIdentifier`
[SWS_DM_00408]	Retrieving data for requested DataIdentifier
[SWS_DM_00412]	Check requested number of DataIdentifiers
[SWS_DM_00413]	Check supported DataIdentifier in active session
[SWS_DM_00414]	Check supported DataIdentifier on active security level
[SWS_DM_00416]	Check supported DataIdentifier in active session
[SWS_DM_00417]	Check supported DataIdentifier on active security level
[SWS_DM_00418]	Writing data for requested DataIdentifier
[SWS_DM_00419]	Reaction on ApplicationError
[SWS_DM_00420]	Instantiation of Diagnostic Server
[SWS_DM_00434]	Providing the `PowerMode` in DoIP protocol messages

Table B.8: Changed Traceables in 18-10

B.3.3 Deleted Traceables in 18-10

Number	Heading
[SWS_DM_00001]	SRS Diagnostics
[SWS_DM_00012]	DoIP configurable source address identification
[SWS_DM_00041]	Behavior according to ISO Multiple client handling flow
[SWS_DM_00043]	Request refusal in case of no resources
[SWS_DM_00044]	Request refusal in case of non-default session active
[SWS_DM_00045]	Ignore ISO same resource access check
[SWS_DM_00048]	Request refusal in case of no resources
[SWS_DM_00051]	Cancellation of Active Protocol with lower priority
[SWS_DM_00052]	Selection between multiple cancellation candidates
[SWS_DM_00066]	Monitor initialization
[SWS_DM_00105]	Configurable Manufacturer Permission Check Services
[SWS_DM_00107]	Configurable Supplier Permission Check Services
[SWS_DM_00118]	Event specific configuration to allow clearing of a DTC
[SWS_DM_00119]	Init value for events with clear allowed information
[SWS_DM_00120]	Description of application interface to control the clear event behavior

▽

△

Number	Heading
[SWS_DM_00125]	Linking between event clear allowed and clearing a DTC
[SWS_DM_00161]	Negative response on not supported GroupOfDTC parameter
[SWS_DM_00166]	Trigger to process event status
[SWS_DM_00180]	Provide Protocol Priority Configurability
[SWS_DM_00182]	Identification of a protocol for Priority Assignment
[SWS_DM_00183]	Wildcards per attribute
[SWS_DM_00184]	Protocol Match Search
[SWS_DM_00185]	No Match
[SWS_DM_00258]	Cancellation of Active Protocol in non-default session
[SWS_DM_00259]	Completion of already Active Protocols in default session
[SWS_DM_00274]	Definition of an active `Diagnostic Protocol`
[SWS_DM_00281]	Cancellation of active DiagnosticConversation in Non-Default Session
[SWS_DM_00282]	Handling of non-/active diagnostic conversations
[SWS_DM_00295]	meta info map vendor type
[SWS_DM_00305]	Const UdsMessage Pointer vendor type
[SWS_DM_00308]	Global Channel Identifier type
[SWS_DM_00316]	Header file
[SWS_DM_00317]	UdsTransportProtocolHandler constructor
[SWS_DM_00318]	UdsTransportProtocolHandler destructor
[SWS_DM_00320]	UdsTransportProtocolHandler UdsTransportProtocolMgr member
[SWS_DM_00321]	constructor member initialization
[SWS_DM_00324]	UdsTransportProtocolHandler UdsTransportProtocolHandlerID member
[SWS_DM_00328]	UdsMessage Pointer vendor type
[SWS_DM_00334]	UdsTransportProtocolMgr may be an abstract class
[SWS_DM_00335]	Header file
[SWS_DM_00339]	ByteVector vendor type
[SWS_DM_00402]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00403]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00405]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00406]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00410]	Check session permission
[SWS_DM_00411]	Check security level permission
[SWS_DM_CONSTR_00207]	Required VINDataIdentifier

Table B.9: Deleted Traceables in 18-10

B.3.4 Added Constraints in 18-10

none

B.3.5 Changed Constraints in 18-10

none

B.3.6 Deleted Constraints in 18-10

none

B.4 Constraint and Specification Item History of this document according to AUTOSAR Release 19-03

B.4.1 Added Traceables in 19-03

Number	Heading
[SWS_DM_00510]	Namespace of Service header files
[SWS_DM_00511]	Implementation Types header files existence
[SWS_DM_00512]	Data Type definitions for AUTOSAR Data Types in Implementation Types header files
[SWS_DM_00513]	Implementation Types header file namespace
[SWS_DM_00526]	
[SWS_DM_00538]	
[SWS_DM_00539]	
[SWS_DM_00540]	
[SWS_DM_00541]	
[SWS_DM_00542]	
[SWS_DM_00543]	
[SWS_DM_00544]	Use of general ara::diag errors
[SWS_DM_00545]	Definition Offer ara::diag errors
[SWS_DM_00546]	Definition Reporting ara::diag errors
[SWS_DM_00547]	Definition UDS NRC ara::diag errors
[SWS_DM_00548]	
[SWS_DM_00549]	
[SWS_DM_00550]	
[SWS_DM_00551]	
[SWS_DM_00552]	
[SWS_DM_00553]	
[SWS_DM_00554]	
[SWS_DM_00555]	

▽

△

Number	Heading
[SWS_DM_00556]	
[SWS_DM_00557]	
[SWS_DM_00558]	
[SWS_DM_00559]	
[SWS_DM_00560]	
[SWS_DM_00561]	Deployment of diagnostic PortInterfaces
[SWS_DM_00562]	Monitor initialization for clearing reason
[SWS_DM_00563]	Monitor initialization for operation cycle restart reason
[SWS_DM_00564]	Monitor initialization for enable condition re-enabling reason
[SWS_DM_00565]	Monitor initialization for DTC setting re-enabling reason
[SWS_DM_00566]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00567]	Ignoring reported events for not started operation cycles
[SWS_DM_00568]	Handling of enable conditions
[SWS_DM_00569]	Handling of storage conditions
[SWS_DM_00570]	Retrieving data for requested DataIdentifier
[SWS_DM_00571]	Reaction on ApplicationError
[SWS_DM_00572]	Writing data for requested DataIdentifier
[SWS_DM_00573]	Reaction on ApplicationError
[SWS_DM_00574]	UDS Service RoutineControl (0x31) startRoutine processing
[SWS_DM_00575]	UDS Service RoutineControl (0x31) requestRoutineResults processing
[SWS_DM_00576]	UDS Service RoutineControl (0x31) stopRoutine processing
[SWS_DM_00577]	Canceling external service processors
[SWS_DM_00578]	
[SWS_DM_00579]	
[SWS_DM_00580]	
[SWS_DM_00581]	
[SWS_DM_00582]	
[SWS_DM_00583]	
[SWS_DM_00584]	
[SWS_DM_00585]	
[SWS_DM_00586]	
[SWS_DM_00587]	
[SWS_DM_00588]	
[SWS_DM_00589]	
[SWS_DM_00590]	
[SWS_DM_00591]	
[SWS_DM_00592]	

▽

△

Number	Heading
[SWS_DM_00593]	
[SWS_DM_00594]	
[SWS_DM_00595]	
[SWS_DM_00596]	
[SWS_DM_00597]	
[SWS_DM_00598]	
[SWS_DM_00599]	
[SWS_DM_00600]	
[SWS_DM_00601]	
[SWS_DM_00602]	
[SWS_DM_00603]	
[SWS_DM_00604]	
[SWS_DM_00605]	
[SWS_DM_00607]	
[SWS_DM_00608]	
[SWS_DM_00609]	
[SWS_DM_00610]	
[SWS_DM_00611]	
[SWS_DM_00612]	
[SWS_DM_00613]	
[SWS_DM_00614]	
[SWS_DM_00615]	
[SWS_DM_00616]	
[SWS_DM_00617]	
[SWS_DM_00618]	
[SWS_DM_00619]	
[SWS_DM_00620]	
[SWS_DM_00634]	
[SWS_DM_00635]	
[SWS_DM_00636]	
[SWS_DM_00637]	
[SWS_DM_00638]	
[SWS_DM_00639]	
[SWS_DM_00640]	
[SWS_DM_00641]	
[SWS_DM_00644]	
[SWS_DM_00646]	

▽

△

Number	Heading
[SWS_DM_00647]	
[SWS_DM_00648]	
[SWS_DM_00649]	
[SWS_DM_00650]	
[SWS_DM_00652]	
[SWS_DM_00653]	
[SWS_DM_00654]	
[SWS_DM_00655]	
[SWS_DM_00656]	
[SWS_DM_00657]	
[SWS_DM_00658]	
[SWS_DM_00663]	
[SWS_DM_00664]	
[SWS_DM_00665]	
[SWS_DM_00666]	
[SWS_DM_00667]	
[SWS_DM_00668]	
[SWS_DM_00669]	
[SWS_DM_00670]	
[SWS_DM_00671]	
[SWS_DM_00672]	
[SWS_DM_00673]	
[SWS_DM_00674]	
[SWS_DM_00691]	
[SWS_DM_00692]	
[SWS_DM_00693]	
[SWS_DM_00694]	
[SWS_DM_00695]	
[SWS_DM_00696]	
[SWS_DM_00697]	
[SWS_DM_00698]	
[SWS_DM_00699]	
[SWS_DM_00700]	
[SWS_DM_00701]	
[SWS_DM_00710]	
[SWS_DM_00711]	
[SWS_DM_00712]	

▽

△

Number	Heading
[SWS_DM_00713]	
[SWS_DM_00714]	
[SWS_DM_00715]	
[SWS_DM_00720]	
[SWS_DM_00721]	
[SWS_DM_00722]	
[SWS_DM_00723]	
[SWS_DM_00724]	
[SWS_DM_00725]	
[SWS_DM_00726]	
[SWS_DM_00731]	
[SWS_DM_00732]	
[SWS_DM_00733]	
[SWS_DM_00734]	
[SWS_DM_00735]	
[SWS_DM_00736]	
[SWS_DM_00740]	
[SWS_DM_00741]	
[SWS_DM_00742]	
[SWS_DM_00743]	
[SWS_DM_00744]	
[SWS_DM_00745]	
[SWS_DM_00750]	
[SWS_DM_00751]	
[SWS_DM_00752]	
[SWS_DM_00753]	
[SWS_DM_00754]	
[SWS_DM_00755]	
[SWS_DM_00756]	
[SWS_DM_00781]	NumberOfStoredEntries
[SWS_DM_00782]	
[SWS_DM_00783]	
[SWS_DM_00784]	
[SWS_DM_00785]	
[SWS_DM_00787]	
[SWS_DM_00788]	
[SWS_DM_00789]	

▽

△

Number	Heading
[SWS_DM_00790]	
[SWS_DM_00791]	
[SWS_DM_00792]	
[SWS_DM_00793]	
[SWS_DM_00794]	
[SWS_DM_00795]	
[SWS_DM_00797]	
[SWS_DM_00798]	
[SWS_DM_00799]	
[SWS_DM_00800]	
[SWS_DM_00801]	
[SWS_DM_00802]	
[SWS_DM_00803]	

Table B.10: Added Traceables in 19-03

B.4.2 Changed Traceables in 19-03

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00042]	Canceling external service processors
[SWS_DM_00058]	DTC interpretation format
[SWS_DM_00064]	Definition of DTC groups
[SWS_DM_00067]	Monitor initialization for clearing reason
[SWS_DM_00068]	Monitor initialization for operation cycle restart reason
[SWS_DM_00069]	Monitor initialization for enable condition re-enabling reason
[SWS_DM_00070]	Monitor initialization for DTC setting re-enabling reason
[SWS_DM_00071]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00106]	Signature of Manufacturer Permission Check Method
[SWS_DM_00108]	Signature of Supplier Permission Check Method
[SWS_DM_00177]	Reaction on ApplicationError
[SWS_DM_00198]	Negative Response processing
[SWS_DM_00199]	Positive Response processing
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00216]	DTC status bit transitions triggered by operation cycle changes

▽

△

Number	Heading
[SWS_DM_00218]	Trip Counter
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00296]	
[SWS_DM_00307]	
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00360]	EcuReset positive response processing after reset
[SWS_DM_00361]	EcuReset application error processing
[SWS_DM_00364]	Negative response processing
[SWS_DM_00366]	Suppression of negative response for functional requests in accordance to ISO 14229-1[1]
[SWS_DM_00367]	No service processing
[SWS_DM_00376]	Positive response processing
[SWS_DM_00382]	Session timeout start
[SWS_DM_00383]	Session timeout stop
[SWS_DM_00384]	
[SWS_DM_00419]	Reaction on ApplicationError
[SWS_DM_00436]	Providing the GID in DoIP protocol messages
[SWS_DM_00479]	Blocking Timer for security access on Restart or Power down - power up cycle
[SWS_DM_00503]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00504]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00505]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00506]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00651]	
[SWS_DM_09017]	
[SWS_DM_CONSTR_00395]	Restriction on DEM-exclusive DiagnosticDataElements

Table B.11: Changed Traceables in 19-03

B.4.3 Deleted Traceables in 19-03

Number	Heading
[SWS_DM_00104]	Supported UDS Services
[SWS_DM_00483]	Cancellation trigger from AAs
[SWS_DM_09028]	

Table B.12: Deleted Traceables in 19-03

B.4.4 Added Constraints in 19-03

none

B.4.5 Changed Constraints in 19-03

none

B.4.6 Deleted Constraints in 19-03

none

B.5 Constraint and Specification Item History of this document according to AUTOSAR Release 19-11

B.5.1 Added Traceables in 19-11

Number	Heading
[SWS_DM_00450]	Security Access subfunction level Permission
[SWS_DM_00502]	Support for Custom Diagnostic Services
[SWS_DM_00642]	
[SWS_DM_00643]	
[SWS_DM_00645]	
[SWS_DM_00659]	
[SWS_DM_00660]	
[SWS_DM_00661]	
[SWS_DM_00662]	
[SWS_DM_00690]	
[SWS_DM_00702]	
[SWS_DM_00730]	
[SWS_DM_00760]	
[SWS_DM_00761]	
[SWS_DM_00762]	
[SWS_DM_00763]	
[SWS_DM_00764]	
[SWS_DM_00765]	
[SWS_DM_00766]	
[SWS_DM_00767]	
[SWS_DM_00770]	
[SWS_DM_00771]	
[SWS_DM_00772]	

▽

△

Number	Heading
[SWS_DM_00773]	
[SWS_DM_00774]	
[SWS_DM_00775]	
[SWS_DM_00776]	
[SWS_DM_00777]	
[SWS_DM_00804]	
[SWS_DM_00805]	
[SWS_DM_00806]	
[SWS_DM_00807]	
[SWS_DM_00808]	
[SWS_DM_00809]	
[SWS_DM_00810]	
[SWS_DM_00811]	Re-enabling of ControlDTCSetting by Diagnostic Application
[SWS_DM_00812]	Re-enabling on transition to default session
[SWS_DM_00813]	Providing the GID in DoIP protocol messages
[SWS_DM_00814]	Providing the PowerMode in DoIP protocol messages
[SWS_DM_00815]	When to send Vehicle announcement messages on interfaces without activation line control
[SWS_DM_00816]	Notification of activation line status change on activation line controlled network interfaces
[SWS_DM_00820]	
[SWS_DM_00821]	
[SWS_DM_00822]	
[SWS_DM_00830]	
[SWS_DM_00831]	
[SWS_DM_00832]	
[SWS_DM_00833]	
[SWS_DM_00834]	
[SWS_DM_00835]	
[SWS_DM_00836]	
[SWS_DM_00837]	
[SWS_DM_00840]	Instantiation of Diagnostic Conversation Interface
[SWS_DM_00841]	Assignment of Diagnostic Conversation to Service Instances
[SWS_DM_00842]	Default session change trigger from AAs
[SWS_DM_00843]	Reset Service Instance fields on end of Diagnostic Conversation
[SWS_DM_00844]	Updating DiagnosticConversation Service Instance fields
[SWS_DM_00845]	Notification about session change
[SWS_DM_00846]	Notification about security-level change

▽

△

Number	Heading
[SWS_DM_00847]	Reinitialization of Service Instance on Cancellation of a Diagnostic Conversation
[SWS_DM_00848]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00849]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00850]	Default Service Interface for reading DiagnosticDataIdentifier
[SWS_DM_00855]	Providing the VIN in DoIP protocol messages
[SWS_DM_00856]	Initial values for Diagnostic Conversation
[SWS_DM_00857]	Signature of Manufacturer Permission Check Method
[SWS_DM_00858]	Signature of Supplier Permission Check Method
[SWS_DM_00859]	Confirmation of service processing
[SWS_DM_00860]	No service processing
[SWS_DM_00861]	Negative response processing
[SWS_DM_00862]	Suppression of negative response for functional requests in accordance to ISO 14229-1[1]
[SWS_DM_00863]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00864]	Checking Supported Subfunction for CompareKey
[SWS_DM_00865]	Communication control service processing
[SWS_DM_00866]	Negative Response processing
[SWS_DM_00867]	UDS service RequestDownload (0x34) processing
[SWS_DM_00868]	UDS service RequestUpload (0x35) processing
[SWS_DM_00869]	UDS service TransferData (0x36) processing
[SWS_DM_00870]	UDS service TransferData (0x36) validation
[SWS_DM_00871]	UDS service RequestTransferExit (0x37) processing
[SWS_DM_00872]	UDS service RequestTransferExit (0x37) validation
[SWS_DM_00873]	Diagnostic event processing interface
[SWS_DM_00874]	Reporting kPrepassed or kPrefailed for events without an assigned debouncing algorithm
[SWS_DM_00875]	Internal debounce counter incrementation
[SWS_DM_00876]	Internal debounce counter decrementation
[SWS_DM_00877]	Starting time-based event debouncing for failed
[SWS_DM_00878]	Starting time-based event debouncing for passed
[SWS_DM_00879]	Application resetting the debounce counter
[SWS_DM_00880]	Debounce time freeze request
[SWS_DM_00881]	Enable condition influence on debouncing behavior (freeze)
[SWS_DM_00882]	Enable condition influence on debouncing behavior (reset)
[SWS_DM_00883]	UDS DTC status bit transitions triggered by test results
[SWS_DM_00884]	Resetting the status of the DTC

▽

△

Number	Heading
[SWS_DM_00885]	UDS DTC status bit transitions triggered by operation cycle changes
[SWS_DM_00886]	Observability of the status byte
[SWS_DM_00887]	Notification about DTC status changes
[SWS_DM_00888]	Observability of indicator status
[SWS_DM_00889]	Automatic starting of operation cycles
[SWS_DM_00890]	Automatic ending of operation cycles
[SWS_DM_00891]	Restart of operation cycles
[SWS_DM_00892]	Operation cycles are only ended once
[SWS_DM_00893]	Triggering for snapshot record storage
[SWS_DM_00894]	Notification event upon snapshot record updates
[SWS_DM_00895]	Triggering for extended data record storage and updates
[SWS_DM_00896]	Handling of DiagnosticClearConditions
[SWS_DM_00897]	Usage of ClearDTC Interface
[SWS_DM_00898]	ClearDTC call on invalid DTC or DTC group
[SWS_DM_00899]	ClearDTC called while another clear operation is in progress
[SWS_DM_00900]	ClearDTC processing in case of memory errors
[SWS_DM_00901]	Possible failure of ClearDTC
[SWS_DM_00902]	NumberOfStoredEntries
[SWS_DM_00903]	Reading DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00904]	Writing DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00905]	Retrieving data for external DiagnosticDataElements
[SWS_DM_00906]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00907]	Default Service Interface for writing DiagnosticDataIdentifier
[SWS_DM_00908]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00909]	Support of Subfunction 0x01 (ON)
[SWS_DM_00910]	Support of Subfunction 0x02 (OFF)
[SWS_DM_00911]	Instances of DTCInformation interface
[SWS_DM_09011]	
[SWS_DM_09013]	
[SWS_DM_09014]	
[SWS_DM_09018]	

Table B.13: Added Traceables in 19-11

B.5.2 Changed Traceables in 19-11

Number	Heading
[SWS_DM_00021]	Direct failed qualification of counter-based events
[SWS_DM_00024]	Qualified failed event using counter-based debouncing
[SWS_DM_00025]	Qualified passed event using counter-based debouncing
[SWS_DM_00029]	Direct passed qualification of counter-based events
[SWS_DM_00032]	Restrictions on restarting a running event debounce timer for failed
[SWS_DM_00033]	Debounce timer behavior upon reported failed
[SWS_DM_00035]	Restrictions on restarting a running event debounce timer for passed
[SWS_DM_00036]	Debounce timer behavior upon reported passed
[SWS_DM_00038]	Continuing a frozen debounce timer
[SWS_DM_00217]	UDS DTC status bit transitions triggered by ClearDiagnosticInformation UDS service
[SWS_DM_00218]	Trip Counter
[SWS_DM_00242]	Re-occurrence after Aging
[SWS_DM_00243]	Aging-related UDS DTC status byte processing
[SWS_DM_00268]	EcuReset positive response processing before reset
[SWS_DM_00279]	Cancellation of a Diagnostic Conversation before Response Transmission
[SWS_DM_00280]	Cancellation of a Diagnostic Conversation at Response Transmission
[SWS_DM_00296]	
[SWS_DM_00307]	
[SWS_DM_00393]	Retrieving data for internal DiagnosticDataElements
[SWS_DM_00401]	Reading Diagnostic Data Identifier on Data Element level
[SWS_DM_00412]	Check requested number of DataIdentifiers
[SWS_DM_00421]	Identification of a Diagnostic Client
[SWS_DM_00425]	Procedure to assign UDS requests to Diagnostic Conversations
[SWS_DM_00426]	Assigning a UDS request to an existing Diagnostic Conversation
[SWS_DM_00427]	Priority of a Diagnostic Conversation
[SWS_DM_00428]	Treatment of priority values
[SWS_DM_00429]	Prioritization in active non-default session
[SWS_DM_00430]	Prioritization against all Diagnostic Conversations
[SWS_DM_00431]	Replacement of Diagnostic Conversations
[SWS_DM_00433]	Refusal of diagnostic request due to busy Diagnostic Conversation
[SWS_DM_00437]	Check supported RoutineIdentifier subfunction on active security level
[SWS_DM_00448]	Check supported RoutineIdentifier subfunction in active session
[SWS_DM_00449]	Supported DoIP message types
[SWS_DM_00475]	DoIP Version
[SWS_DM_00478]	Persistent Storage of failed attempts to change security level
[SWS_DM_00479]	Blocking Timer for security access on Restart or Power down - power up cycle

△

Number	Heading
[SWS_DM_00482]	Cancellation of a Diagnostic Conversation
[SWS_DM_00507]	Length check on UDS Service 0x27 request with Subfunction for Request-Seed
[SWS_DM_00526]	
[SWS_DM_00538]	
[SWS_DM_00539]	
[SWS_DM_00540]	
[SWS_DM_00541]	
[SWS_DM_00542]	
[SWS_DM_00543]	
[SWS_DM_00548]	
[SWS_DM_00549]	
[SWS_DM_00550]	
[SWS_DM_00551]	
[SWS_DM_00552]	
[SWS_DM_00553]	
[SWS_DM_00554]	
[SWS_DM_00555]	
[SWS_DM_00556]	
[SWS_DM_00557]	
[SWS_DM_00559]	
[SWS_DM_00560]	
[SWS_DM_00562]	Monitor initialization for clearing reason
[SWS_DM_00563]	Monitor initialization for operation cycle restart reason
[SWS_DM_00564]	Monitor initialization for enable condition re-enabling reason
[SWS_DM_00565]	Monitor initialization for DTC setting re-enabling reason
[SWS_DM_00567]	Ignoring reported events for not started operation cycles
[SWS_DM_00568]	Handling of enable conditions
[SWS_DM_00570]	Retrieving data for requested DataIdentifier
[SWS_DM_00571]	Reaction on ApplicationError
[SWS_DM_00572]	Writing data for requested DataIdentifier
[SWS_DM_00573]	Reaction on ApplicationError
[SWS_DM_00574]	UDS Service RoutineControl (0x31) startRoutine processing
[SWS_DM_00575]	UDS Service RoutineControl (0x31) requestRoutineResults processing
[SWS_DM_00576]	UDS Service RoutineControl (0x31) stopRoutine processing
[SWS_DM_00584]	
[SWS_DM_00585]	
[SWS_DM_00586]	

▽

△

Number	Heading
[SWS_DM_00587]	
[SWS_DM_00588]	
[SWS_DM_00589]	
[SWS_DM_00590]	
[SWS_DM_00591]	
[SWS_DM_00592]	
[SWS_DM_00593]	
[SWS_DM_00594]	
[SWS_DM_00596]	
[SWS_DM_00597]	
[SWS_DM_00598]	
[SWS_DM_00599]	
[SWS_DM_00601]	
[SWS_DM_00603]	
[SWS_DM_00604]	
[SWS_DM_00605]	
[SWS_DM_00616]	
[SWS_DM_00618]	
[SWS_DM_00634]	
[SWS_DM_00635]	
[SWS_DM_00636]	
[SWS_DM_00637]	
[SWS_DM_00638]	
[SWS_DM_00640]	
[SWS_DM_00644]	
[SWS_DM_00646]	
[SWS_DM_00647]	
[SWS_DM_00648]	
[SWS_DM_00649]	
[SWS_DM_00650]	
[SWS_DM_00651]	
[SWS_DM_00652]	
[SWS_DM_00653]	
[SWS_DM_00654]	
[SWS_DM_00655]	
[SWS_DM_00656]	
[SWS_DM_00657]	

▽

△

Number	Heading
[SWS_DM_00658]	
[SWS_DM_00663]	
[SWS_DM_00664]	
[SWS_DM_00665]	
[SWS_DM_00666]	
[SWS_DM_00667]	
[SWS_DM_00668]	
[SWS_DM_00669]	
[SWS_DM_00670]	
[SWS_DM_00671]	
[SWS_DM_00672]	
[SWS_DM_00673]	
[SWS_DM_00674]	
[SWS_DM_00692]	
[SWS_DM_00694]	
[SWS_DM_00695]	
[SWS_DM_00696]	
[SWS_DM_00697]	
[SWS_DM_00698]	
[SWS_DM_00699]	
[SWS_DM_00700]	
[SWS_DM_00701]	
[SWS_DM_00712]	
[SWS_DM_00713]	
[SWS_DM_00714]	
[SWS_DM_00715]	
[SWS_DM_00720]	
[SWS_DM_00721]	
[SWS_DM_00722]	
[SWS_DM_00723]	
[SWS_DM_00724]	
[SWS_DM_00725]	
[SWS_DM_00726]	
[SWS_DM_00731]	
[SWS_DM_00732]	
[SWS_DM_00733]	
[SWS_DM_00734]	

▽

△

Number	Heading
[SWS_DM_00735]	
[SWS_DM_00736]	
[SWS_DM_00740]	
[SWS_DM_00741]	
[SWS_DM_00742]	
[SWS_DM_00743]	
[SWS_DM_00744]	
[SWS_DM_00745]	
[SWS_DM_00750]	
[SWS_DM_00751]	
[SWS_DM_00752]	
[SWS_DM_00753]	
[SWS_DM_00754]	
[SWS_DM_00755]	
[SWS_DM_00756]	
[SWS_DM_00782]	
[SWS_DM_00783]	
[SWS_DM_00787]	
[SWS_DM_00788]	
[SWS_DM_00789]	
[SWS_DM_00790]	
[SWS_DM_00791]	
[SWS_DM_00792]	
[SWS_DM_00797]	
[SWS_DM_00798]	
[SWS_DM_00799]	
[SWS_DM_00800]	
[SWS_DM_00801]	
[SWS_DM_00802]	
[SWS_DM_09012]	
[SWS_DM_09017]	

Table B.14: Changed Traceables in 19-11

B.5.3 Deleted Traceables in 19-11

Number	Heading
[SWS_DM_00002]	Automatic starting of operation cycles
[SWS_DM_00003]	Automatic ending of operation cycles
[SWS_DM_00008]	Diagnostic event processing interface
[SWS_DM_00011]	Selectability of parallelism mode
[SWS_DM_00016]	Configurable number of supported parallel Diagnostic Conversations
[SWS_DM_00019]	Internal debounce counter incrementation
[SWS_DM_00020]	Internal debounce counter decrementation
[SWS_DM_00026]	Application resetting the debounce counter
[SWS_DM_00031]	Starting time-based event debouncing for failed
[SWS_DM_00034]	Starting time-based event debouncing for passed
[SWS_DM_00037]	Debounce time freeze request
[SWS_DM_00042]	Canceling external service processors
[SWS_DM_00067]	Monitor initialization for clearing reason
[SWS_DM_00068]	Monitor initialization for operation cycle restart reason
[SWS_DM_00069]	Monitor initialization for enable condition re-enabling reason
[SWS_DM_00070]	Monitor initialization for DTC setting re-enabling reason
[SWS_DM_00071]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00074]	Handling of enable conditions
[SWS_DM_00087]	Enable condition influence on debouncing behavior (freeze)
[SWS_DM_00089]	Reporting kPrepassed or kPrefailed for events without an assigned debouncing algorithm
[SWS_DM_00106]	Signature of Manufacturer Permission Check Method
[SWS_DM_00108]	Signature of Supplier Permission Check Method
[SWS_DM_00131]	UDS service RequestDownload (0x34) processing
[SWS_DM_00136]	UDS service RequestUpload (0x35) processing
[SWS_DM_00138]	UDS service TransferData (0x36) processing
[SWS_DM_00139]	UDS service TransferData (0x36) validation
[SWS_DM_00142]	UDS service RequestTransferExit (0x37) processing
[SWS_DM_00143]	UDS service RequestTransferExit (0x37) validation
[SWS_DM_00153]	Triggering for snapshot record storage
[SWS_DM_00156]	Triggering for extended data record storage and updates
[SWS_DM_00167]	Ignoring reported events for not started operation cycles
[SWS_DM_00168]	Availability of DiagnosticMonitor service interfaces
[SWS_DM_00169]	Restart of operation cycles
[SWS_DM_00177]	Reaction on ApplicationError
[SWS_DM_00192]	Operation cycles are only ended once
[SWS_DM_00197]	Communication control service processing

▽

△

Number	Heading
[SWS_DM_00198]	Negative Response processing
[SWS_DM_00205]	Providing the VIN in DoIP protocol messages
[SWS_DM_00210]	UDS Service RoutineControl (0x31) startRoutine processing
[SWS_DM_00211]	UDS Service RoutineControl (0x31) requestRoutineResults processing
[SWS_DM_00212]	UDS Service RoutineControl (0x31) stopRoutine processing
[SWS_DM_00214]	DTC status bit transitions triggered by test results
[SWS_DM_00215]	Resetting the status of the DTC
[SWS_DM_00216]	DTC status bit transitions triggered by operation cycle changes
[SWS_DM_00219]	Observability of the status byte
[SWS_DM_00220]	Notification about DTC status changes
[SWS_DM_00222]	Observability of indicator status
[SWS_DM_00232]	Support of Subfunction 0x01 (ON)
[SWS_DM_00233]	Support of Subfunction 0x02 (OFF)
[SWS_DM_00248]	Notification about session change
[SWS_DM_00249]	Checking Supported Subfunction for RequestSeed
[SWS_DM_00250]	Notification about security-level change
[SWS_DM_00260]	instances of interface ClearDTC
[SWS_DM_00261]	Usage of ClearDTC Interface
[SWS_DM_00263]	ClearDTC call on invalid DTC or DTCgroup
[SWS_DM_00265]	ClearDTC called while another clear operation is in progress
[SWS_DM_00266]	ClearDTC processing in case of memory errors
[SWS_DM_00267]	Possible failure of ClearDTC
[SWS_DM_00273]	Notification event upon snapshot record updates
[SWS_DM_00341]	Confirmation of service processing
[SWS_DM_00362]	Checking Supported Subfunction for CompareKey
[SWS_DM_00364]	Negative response processing
[SWS_DM_00366]	Suppression of negative response for functional requests in accordance to ISO 14229-1[1]
[SWS_DM_00367]	No service processing
[SWS_DM_00377]	Enable condition influence on debouncing behavior (reset)
[SWS_DM_00379]	Handling of storage conditions
[SWS_DM_00397]	Retrieving data for external DiagnosticDataElements
[SWS_DM_00404]	Default Service Interface for reading DiagnosticDataIdentifier
[SWS_DM_00407]	Default Service Interface for writing DiagnosticDataIdentifier
[SWS_DM_00408]	Retrieving data for requested DataIdentifier
[SWS_DM_00418]	Writing data for requested DataIdentifier
[SWS_DM_00419]	Reaction on ApplicationError

▽

△

Number	Heading
[SWS_DM_00422]	Instantiation of Diagnostic Conversation Service Interface
[SWS_DM_00423]	Assignment of Diagnostic Conversation to Service Instances
[SWS_DM_00424]	Reset Service Instance fields on end of Diagnostic Conversation
[SWS_DM_00432]	Initial values for Diagnostic Conversation
[SWS_DM_00434]	Providing the PowerMode in DoIP protocol messages
[SWS_DM_00435]	Default session change trigger from AAs
[SWS_DM_00436]	Providing the GID in DoIP protocol messages
[SWS_DM_00476]	User Controlled Warning IndicatorRequest-bit
[SWS_DM_00477]	Not Storing of 'warningIndicatorRequested' bit
[SWS_DM_00481]	Handling of DiagnosticClearConditions
[SWS_DM_00484]	Updating DiagnosticConversation Service Instance fields
[SWS_DM_00485]	Reinitialization of Service Instance on Cancellation of a Diagnostic Conversation
[SWS_DM_00503]	Reading Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00504]	Reading Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00505]	Writing Diagnostic Data Identifier by DataIdentifier interface
[SWS_DM_00506]	Writing Diagnostic Data Identifier by GenericUDSService interface
[SWS_DM_00508]	Reading DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00509]	Writing DiagnosticDataIdentifier configured for representing VIN
[SWS_DM_00566]	Monitor initialization for storage condition reenabling reason
[SWS_DM_00569]	Handling of storage conditions
[SWS_DM_00781]	NumberOfStoredEntries

Table B.15: Deleted Traceables in 19-11

B.5.4 Added Constraints in 19-11

none

B.5.5 Changed Constraints in 19-11

none

B.5.6 Deleted Constraints in 19-11

none