Document Title	Specification of Diagnostic Log and Trace
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	351
Document Classification	Standard

Document Version	1.4.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	1.4.0	AUTOSAR Release Management	• Changed SWS_Dlt_00477
31.10.2013	1.3.1	AUTOSAR Release Management	• Minor corrections • Editorial changes • Removed chapter(s) on change documentation
26.02.2013	1.3.0	AUTOSAR Administration	• Modeling of Services: introduction of formal descriptions of service interfaces • Reworked according to the new SWS_BSWGeneral
09.12.2011	1.2.0	AUTOSAR Administration	• Added Dlt control messages for getting values of modifiable parameters • • Modification and update of Dem and Dcm interfaces • • Added FIBEX example for non verbose transmission mode
21.10.2010	1.1.0	AUTOSAR Administration	• Bug fixes and extension of Dlt control message specification • Update of communication with Dem (Dem_GetEventFreezeFrameData) • Update of interface to Dcm (Dlt_ReadData)
30.11.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview 7

2 Acronyms and abbreviations 9

3 Related documentation 10

3.1 Input documents 10
3.2 Related standards and norms 11
3.3 Related Specification 11

4 Constraints and assumptions 12

4.1 Assumptions 12
4.1.1 Available RAM 12
4.1.2 Available NVRAM 12
4.1.3 Communication interface 12
4.2 Limitations 12
4.2.1 Runtime resources 12
4.2.2 VFB Trace 12
4.2.3 Security 13
4.2.4 Dlt communication module 13
4.3 Applicability to car domains 13

5 Dependencies to other modules 14

5.1 File structure 14
5.1.1 Code file structure 14
5.1.2 Header file structure 15

6 Requirements traceability 16

7 Functional specification 29

7.1 Dlt term definition 29
7.1.1 Log and trace message 29
7.1.2 User 29
7.1.3 Log 29
7.1.4 Trace 29
7.1.5 ECU ID 29
7.1.6 Session ID 30
7.1.7 Application ID 30
7.1.8 Context ID 30
7.1.9 Message ID 30
7.1.10 Log level and trace status 30
7.1.11 Time 31
7.1.12 Payload 31
7.1.13 External client 31
7.2 Use Cases for logging and tracing with Dlt 31
7.2.1 Use Case general logging with Dlt 31
7.2.2 Use Case logging over UDS with Dlt 32
7.2.3 Use Case tracing of VFB 33
7.2.4 Use Case runtime configuration of Dlt 34
7.2.5 Use Case Dlt interaction only over Dlt communication module 35

7.3 Internal behavior of Dlt Module 35
7.3.1 Overview 35
7.3.2 Startup and Shutdown behavior 37
7.3.3 Communication with producer of log and trace messages 38
7.3.4 Recommendation for generation of Message IDs 49
7.4 Communication from Dlt with external client 49
7.4.1 Communication over standard Dcm channel 49
7.4.2 Communication over Dlt Communication Module 51
7.5 Security 52
7.5.1 Securing communication over Dcm 52
7.5.2 Security for communication over Dlt communication module 53
7.6 Runtime management and Implementation 53
7.6.1 Buffering Messages 53
7.6.2 Bandwidth management 54
7.6.3 Interfaces and behavior of Dlt communication module 55
7.6.4 Administration of pairs of Application ID and Context ID, log levels and trace status 55
7.6.5 Message Filtering 63
7.6.6 Storing Configuration in NVRAM 64
7.6.7 Processing of control messages 66
7.6.8 Message Handling 66
7.7 Protocol Specification (for transmitting to a external client and saving on the client) 68
7.7.1 Dlt Message Format in General 69
7.7.2 Header Definition of the Dlt Protocol 69
7.7.3 Standard Header 69
7.7.4 Dlt Extended Header 75
7.7.5 Payload 81
7.7.6 Additional Message Parts 101
7.7.7 Predefined messages 102
7.7.8 Connection management 123
7.8 Error classification 124
7.9 Scheduling strategy 125

8 API specification 126

8.1 Overview 126
8.1.1 Imported types 127
8.2 Service Interfaces 128
8.2.1 Client-Server-Interfaces 128
8.2.2 Ports 135
8.3 Type definitions 136
8.3.1 Dlt_ConfigType 136
8.3.2 Dlt_MessageTypeType 137
8.3.3 Dlt_SessionIDType 137
8.3.4 Dlt_ApplicationIDType 137
8.3.5 Dlt_ContextIDType 137
8.3.6 Dlt_MessageIDType 138
8.3.7 Dlt_MessageOptionsType 138
8.3.8 Dlt_MessageLogLevelType 138
8.3.9 Dlt_MessageTraceType 138

8.3.10 Dlt_MessageControlInfoType 139
8.3.11 Dlt_MessageNetworkTraceInfoType 139
8.3.12 Dlt_ MessageArgumentCount 139
8.3.13 Dlt_MessageLogInfoType 139
8.3.14 Dlt_MessageTraceInfoType 140
8.3.15 Dlt_ReturnType 140
8.4 Function definitions 140
8.4.1 General provided Functions for BSW-modules 141
8.4.2 Provided functions for sending log messages from SW-Cs 141
8.4.3 Provided function for fan-out capability of Det 144
8.4.4 Provided interfaces for Dcm 145
8.4.5 Interfaces provided by Dlt core module for internal use with Dlt communication module 148
8.5 Configurable interfaces 149
8.5.1 Expected Interfaces from SW-Cs 149
8.6 Expected Interfaces 152
8.6.1 Expected Interfaces from Dcm 152
8.6.2 Expected Interfaces from Dlt communication module 152
8.6.3 Expected Interfaces from Gpt 157
9 Sequence diagrams 158
9.1 Dlt initialization 158
9.2 General logging with Dlt 159
9.3 Logging over UDS by using the Dcm interfaces 160
9.4 Tracing of VFB 162
9.5 Runtime configuration of Dlt 163
9.6 Dlt interaction only over Dlt communication module 164
9.7 Dlt interaction with Dem 165
9.8 Dlt interaction with Gpt 166
10 Configuration specification 167
10.1 How to read this chapter 167
10.2 Containers and configuration parameters 168
10.2.1 Dlt 168
10.2.2 DltGeneral 169
10.2.3 DltMemory 172
10.2.4 DltVfbTrace 174
10.2.5 DltMultipleConfigurationContainer 175
10.2.6 DltBandwidth 176
10.2.7 DltMessageFiltering 177
10.2.8 DltProtocol 179
10.3 Published Information 181
11 Not applicable requirements 182

Known Limitations

The Dlt module is not prepared for multi-core systems. It works with a centralized buffer for storing messages if the Dlt module is not initialized. Hence, if data is

handed to the Dlt module spontaneously from different cores in parallel, the unsynchronized access to the buffer will fail.

Therefore, code running on a core other than the main core must not spontaneously hand data to the dlt module. Configuration must be done accordingly or the usage of this buffer for storing messages at startup must be disabled.
Users of Dlt module which may be affected on multi-core systems are the RTE (VFB tracing) and SWCs.

1 Introduction and functional overview

Dlt provides a generic Logging and Tracing functionality for SW-Cs and the BSW modules RTE, Det and Dem. Main focuses of this document are to specify the container, how data is buffered locally and exported over a communication interface.
The following figure shows, how Dlt is integrated into the AUTOSAR architecture and how the main functionality of Dlt shall be realized.

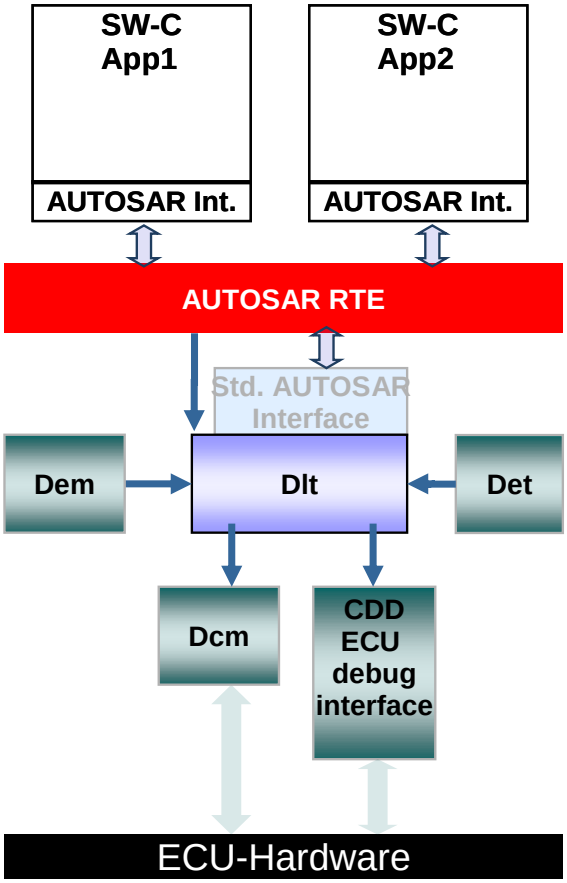


Figure 2 The position of Dlt in the AUTOSAR layered architecture

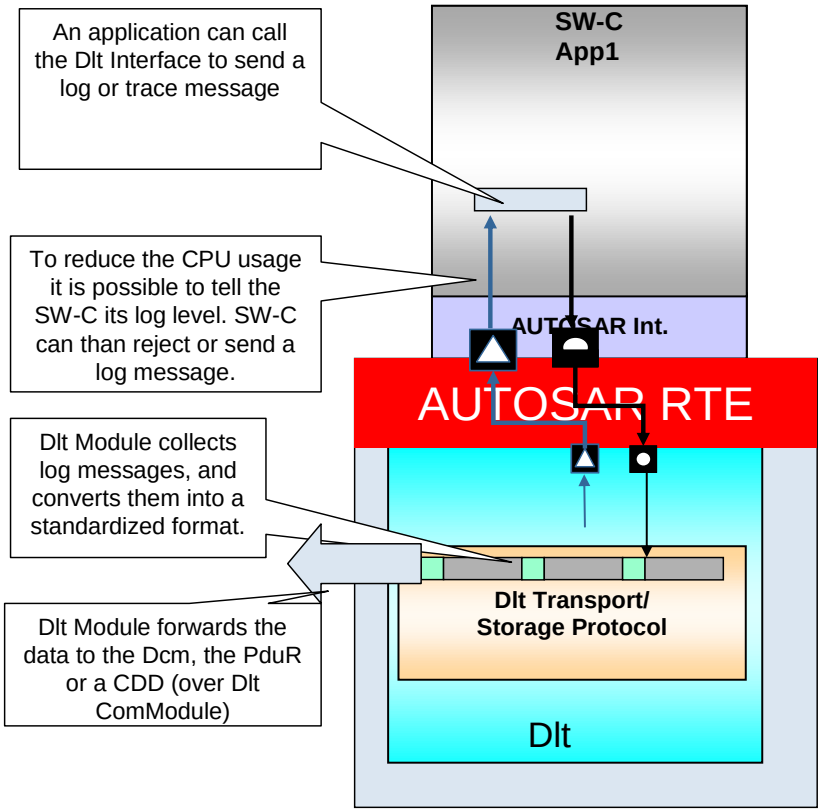


Figure 3 General workflow for sending log and trace messages over Dlt module

Dlt provide the following functionalities:

- Logging
  - Logging of errors, warnings and info messages from AUTOSAR SW-Cs, providing a standardized AUTOSAR interface
  - Gather all log and trace messages from all AUTOSAR SW-Cs in a centralized AUTOSAR service component (Dlt) in BSW
  - Log messages from Det
  - Log messages from Dem
- Tracing
  - Trace RTE/VFB
- Control
  - Enable/disable individual log and trace messages
  - Control trace levels individually by back channel
- Generic
  - Dlt available during debugging and production phase
  - Access over standard diagnosis or platform specific test interface
  - Security mechanisms to prevent misuse in production phase

2 Acronyms and abbreviations

Abbreviation / Acronym:	Description:
API	Application programming interface
Dlt	Diagnostic Log and Trace
LSB	Least Significant Bit
MSB	Most Significant Bit
OBD	On-Board-Diagnose
ODX	Open Diagnostic Data Exchange
ROE	Respond On Event
TCP/IP	Transmission Control Protocol/Internet Protocol
USB	Universal Serial Bus
XCP	Universal Measurement and Calibration Protocol

3 Related documentation

3.1 Input documents

[1] General Requirements on Basic Software Modules,
AUTOSAR_SRS_BSWGeneral.pdf

[2] Specification of PDU Router,
AUTOSAR_SWS_PDURouter.pdf

[3] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[4] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[5] Specification of Diagnostic Event Manager,
AUTOSAR_SWS_DiagnosticEventManager.pdf

[6] Specification of Diagnostic Communication Manager
AUTOSAR_SWS_DiagnosticCommunicationManager.pdf

[7] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[8] Software Component Template,
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[9] Specification of RTE Software,
AUTOSAR_SWS_RTE.pdf

[10] AUTOSAR Virtual Functional Bus,
AUTOSAR_EXP_VFB.pdf

[11] Specification of Development Error Tracer,
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[12] Specification of NVRAM Manager,
AUTOSAR_SWS_NVRAMManager.pdf

[13] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[14] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[15] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

[i] ISO 14229, Road vehicles — Unified diagnostic services (UDS) — Specification and requirements , ISO, second edition 2006-12-01

[ii] ISO/IEC 9899 Programming languages – C

[iii] ISO 22901, Road vehicles -- Open diagnostic data exchange (ODX) – ODX-Standard (ASAM MCD-2D))

3.3 Related Specification

AUTOSAR provides a General Specification on Basic Software modules [15] (SWS BSW General), which is also valid for DLT.
Thus, the specification SWS BSW General shall be considered as additional and required specification for DLT.

4 Constraints and assumptions

4.1 Assumptions

4.1.1 Available RAM

The ECU shall provide enough RAM to buffer temporarily the log and trace messages. The needed amount of memory depends on the number of log and trace message sources, the amount of data and the limited speed of the communication interface. During startup time, enough memory shall be available to store the data, until the log and trace external client is connected. If the available memory is too small, some log and trace messages may be lost.

Additionally the ECU shall provide memory for the log level table. The size depends on the amount of different Application IDs and Context IDs using Dlt.

4.1.2 Available NVRAM

The ECU shall provide enough NVRAM to store persistently the log level table. The size depends on the amount of different Application IDs and Context IDs using Dlt.

4.1.3 Communication interface

Dlt needs at least one communication interface to communicate with an external client. For security reasons the UDS communication interface of the Dcm can be used. Then the interface between Dcm and Dlt shall be used. For higher bandwidth e.g. CAN, FlexRay or Ethernet can be interfaced directly over the PDU router.

4.2 Limitations

4.2.1 Runtime resources

Dlt needs a small amount of runtime resources, even if all sources of log and trace messages are disabled. Only a single condition shall be checked for each log and trace message source. If sources of log and trace messages are enabled, it depends on the number of enabled sources, how much runtime resource is needed. Main runtime resource is needed for communication purposes.

4.2.2 VFB Trace

As VFB trace can produce a lot of traffic, this source shall be used very carefully. Only these values and function calls shall be traced, which are useful for analyzing

malfunction or value changes. Even the frequency of calls shall be regarded, not to produce too much traffic.

4.2.3 Security

An external client must be connected to change the log levels and to communicate over a diagnostic channel.

4.2.4 Dlt communication module

Dlt does not define a specific communication interface. The Dlt specification defines an API to an internal Dlt communication module. It is up to the implementer, how this communication module is implemented and how it communicates with a possible CD (e.g. Serial or USB) or the PDU Router (e.g. CAN, FlexRay, Ethernet).

4.3 Applicability to car domains

This basic software module can be used in all car domains.

5 Dependencies to other modules

This section describes the relationship with other modules within the basic software. It describes the services that are used by these modules.

The Dlt module has dependencies to the following other AUTOSAR modules:

Det
Det has to forward all calls to *Det_ReportError()* to Dlt.

Dem
Dem forwards all calls to *Dem_SetEventStatus() and Dem_ResetEventStatus()* to Dlt.

Dcm
Dlt uses Dcm as a communication interface. The diagnostic services *ReadDataByIdentifier()*, *WriteDataByIdentifier()* and *ResponseOnEvent()* are used. Dcm provides an Interface to be used by Dlt. Dlt uses the security mechanisms of Dcm.

NVRAM-Manager
Dlt uses the NVRAM-Manager to store data persistently. NVRAM-Blocks are assigned to Dlt. In this blocks Dlt stores some data persistently.

RTE/VFB-Trace
Dlt traces RTE events. RTE provides Macros, which are implemented in Dlt, comparable to the VFB-Trace. Dlt decides which Macros to be implemented during configuration time and decides during runtime which traces are generated. Detailed description can be found in chapter 7.3.3.2.

SW-C
Dlt provides an interface for SW-Cs to generate log and trace messages. SW-C shall provide a client-server interface to be able to set log levels by Dlt.

5.1 File structure

5.1.1 Code file structure

[SWS_Dlt_00482] ⌈The module header file Dlt.h shall include Rte_Dlt_Type.h to include the types which are common used by BSW Modules and Software Components. Dlt.h and all Dlt*cfg.h files shall only contain types, that are not already defined in Rte_Dlt_Type.h. ⌋ ()

5.1.2 Header file structure

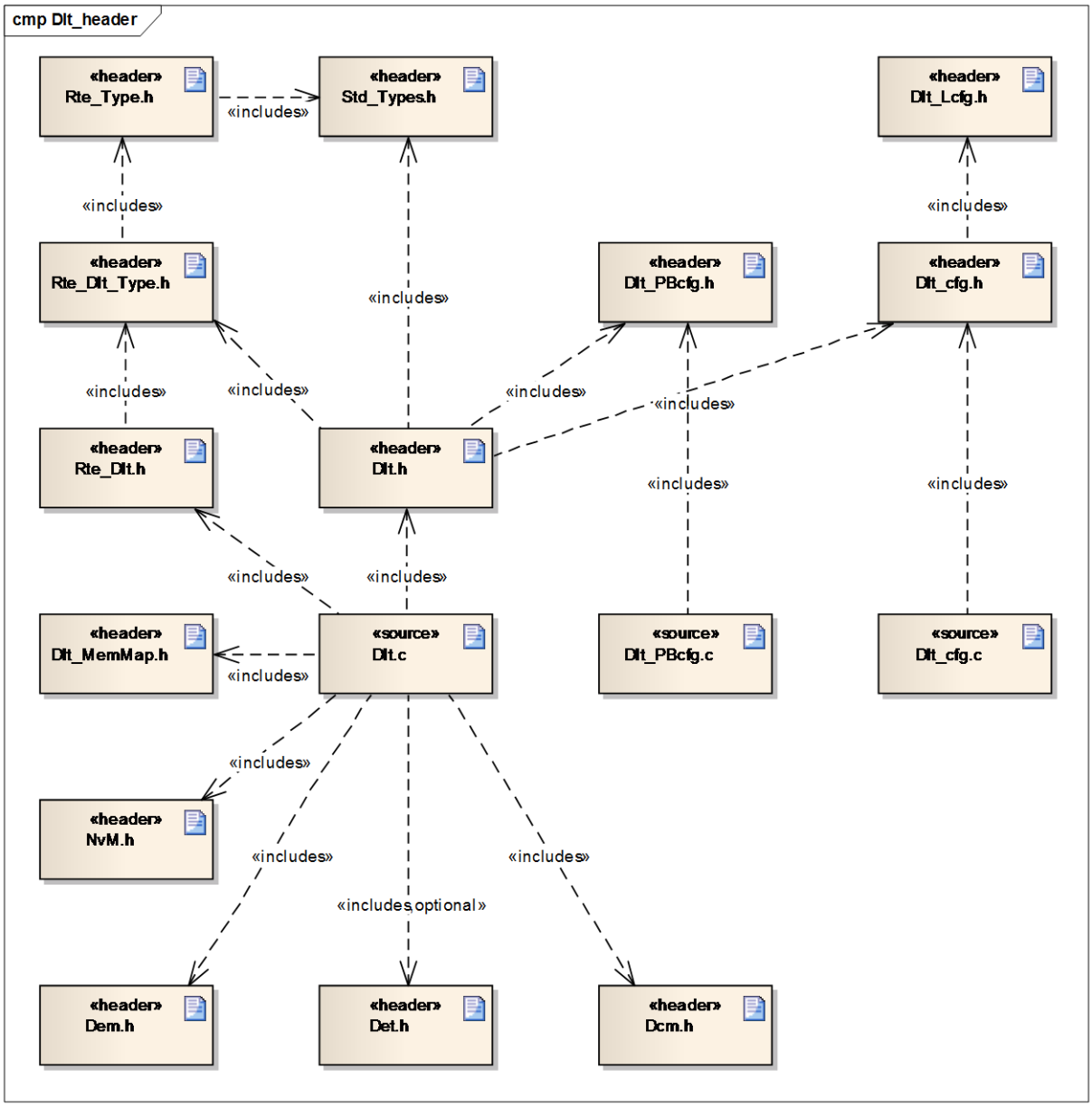


Figure 4 Header file structure recommended for Dlt source

6 Requirements traceability

Requirement	Description	Satisfied by
-	-	SWS_Dlt_00005
-	-	SWS_Dlt_00009
-	-	SWS_Dlt_00010
-	-	SWS_Dlt_00011
-	-	SWS_Dlt_00014
-	-	SWS_Dlt_00015
-	-	SWS_Dlt_00016
-	-	SWS_Dlt_00017
-	-	SWS_Dlt_00018
-	-	SWS_Dlt_00019
-	-	SWS_Dlt_00022
-	-	SWS_Dlt_00023
-	-	SWS_Dlt_00024
-	-	SWS_Dlt_00026
-	-	SWS_Dlt_00027
-	-	SWS_Dlt_00031
-	-	SWS_Dlt_00049
-	-	SWS_Dlt_00050
-	-	SWS_Dlt_00051
-	-	SWS_Dlt_00053
-	-	SWS_Dlt_00057
-	-	SWS_Dlt_00060
-	-	SWS_Dlt_00061
-	-	SWS_Dlt_00066
-	-	SWS_Dlt_00070
-	-	SWS_Dlt_00072
-	-	SWS_Dlt_00078
-	-	SWS_Dlt_00080
-	-	SWS_Dlt_00081
-	-	SWS_Dlt_00082
-	-	SWS_Dlt_00083
-	-	SWS_Dlt_00085
-	-	SWS_Dlt_00087
-	-	SWS_Dlt_00089
-	-	SWS_Dlt_00090
-	-	SWS_Dlt_00091

-	-	SWS_Dlt_00094
-	-	SWS_Dlt_00095
-	-	SWS_Dlt_00103
-	-	SWS_Dlt_00104
-	-	SWS_Dlt_00107
-	-	SWS_Dlt_00108
-	-	SWS_Dlt_00113
-	-	SWS_Dlt_00118
-	-	SWS_Dlt_00120
-	-	SWS_Dlt_00121
-	-	SWS_Dlt_00123
-	-	SWS_Dlt_00124
-	-	SWS_Dlt_00125
-	-	SWS_Dlt_00126
-	-	SWS_Dlt_00129
-	-	SWS_Dlt_00134
-	-	SWS_Dlt_00139
-	-	SWS_Dlt_00145
-	-	SWS_Dlt_00147
-	-	SWS_Dlt_00148
-	-	SWS_Dlt_00149
-	-	SWS_Dlt_00150
-	-	SWS_Dlt_00152
-	-	SWS_Dlt_00153
-	-	SWS_Dlt_00155
-	-	SWS_Dlt_00156
-	-	SWS_Dlt_00157
-	-	SWS_Dlt_00160
-	-	SWS_Dlt_00161
-	-	SWS_Dlt_00169
-	-	SWS_Dlt_00170
-	-	SWS_Dlt_00171
-	-	SWS_Dlt_00172
-	-	SWS_Dlt_00173
-	-	SWS_Dlt_00175
-	-	SWS_Dlt_00176
-	-	SWS_Dlt_00177
-	-	SWS_Dlt_00182
-	-	SWS_Dlt_00183

-	-	SWS_Dlt_00186
-	-	SWS_Dlt_00188
-	-	SWS_Dlt_00189
-	-	SWS_Dlt_00190
-	-	SWS_Dlt_00191
-	-	SWS_Dlt_00192
-	-	SWS_Dlt_00193
-	-	SWS_Dlt_00199
-	-	SWS_Dlt_00200
-	-	SWS_Dlt_00201
-	-	SWS_Dlt_00203
-	-	SWS_Dlt_00204
-	-	SWS_Dlt_00205
-	-	SWS_Dlt_00206
-	-	SWS_Dlt_00209
-	-	SWS_Dlt_00210
-	-	SWS_Dlt_00211
-	-	SWS_Dlt_00212
-	-	SWS_Dlt_00213
-	-	SWS_Dlt_00214
-	-	SWS_Dlt_00215
-	-	SWS_Dlt_00216
-	-	SWS_Dlt_00217
-	-	SWS_Dlt_00218
-	-	SWS_Dlt_00219
-	-	SWS_Dlt_00220
-	-	SWS_Dlt_00222
-	-	SWS_Dlt_00224
-	-	SWS_Dlt_00225
-	-	SWS_Dlt_00226
-	-	SWS_Dlt_00227
-	-	SWS_Dlt_00228
-	-	SWS_Dlt_00229
-	-	SWS_Dlt_00230
-	-	SWS_Dlt_00231
-	-	SWS_Dlt_00232
-	-	SWS_Dlt_00233
-	-	SWS_Dlt_00235
-	-	SWS_Dlt_00236

-	-	SWS_Dlt_00237
-	-	SWS_Dlt_00242
-	-	SWS_Dlt_00244
-	-	SWS_Dlt_00247
-	-	SWS_Dlt_00248
-	-	SWS_Dlt_00249
-	-	SWS_Dlt_00253
-	-	SWS_Dlt_00255
-	-	SWS_Dlt_00256
-	-	SWS_Dlt_00257
-	-	SWS_Dlt_00258
-	-	SWS_Dlt_00259
-	-	SWS_Dlt_00260
-	-	SWS_Dlt_00261
-	-	SWS_Dlt_00262
-	-	SWS_Dlt_00278
-	-	SWS_Dlt_00279
-	-	SWS_Dlt_00280
-	-	SWS_Dlt_00281
-	-	SWS_Dlt_00283
-	-	SWS_Dlt_00292
-	-	SWS_Dlt_00295
-	-	SWS_Dlt_00296
-	-	SWS_Dlt_00297
-	-	SWS_Dlt_00298
-	-	SWS_Dlt_00299
-	-	SWS_Dlt_00300
-	-	SWS_Dlt_00305
-	-	SWS_Dlt_00306
-	-	SWS_Dlt_00307
-	-	SWS_Dlt_00308
-	-	SWS_Dlt_00309
-	-	SWS_Dlt_00312
-	-	SWS_Dlt_00313
-	-	SWS_Dlt_00318
-	-	SWS_Dlt_00320
-	-	SWS_Dlt_00321
-	-	SWS_Dlt_00324
-	-	SWS_Dlt_00325

-	-	SWS_Dlt_00326
-	-	SWS_Dlt_00329
-	-	SWS_Dlt_00331
-	-	SWS_Dlt_00332
-	-	SWS_Dlt_00334
-	-	SWS_Dlt_00335
-	-	SWS_Dlt_00336
-	-	SWS_Dlt_00337
-	-	SWS_Dlt_00338
-	-	SWS_Dlt_00348
-	-	SWS_Dlt_00350
-	-	SWS_Dlt_00351
-	-	SWS_Dlt_00353
-	-	SWS_Dlt_00354
-	-	SWS_Dlt_00355
-	-	SWS_Dlt_00356
-	-	SWS_Dlt_00357
-	-	SWS_Dlt_00358
-	-	SWS_Dlt_00362
-	-	SWS_Dlt_00363
-	-	SWS_Dlt_00364
-	-	SWS_Dlt_00366
-	-	SWS_Dlt_00367
-	-	SWS_Dlt_00369
-	-	SWS_Dlt_00370
-	-	SWS_Dlt_00371
-	-	SWS_Dlt_00372
-	-	SWS_Dlt_00373
-	-	SWS_Dlt_00374
-	-	SWS_Dlt_00375
-	-	SWS_Dlt_00376
-	-	SWS_Dlt_00377
-	-	SWS_Dlt_00382
-	-	SWS_Dlt_00385
-	-	SWS_Dlt_00386
-	-	SWS_Dlt_00387
-	-	SWS_Dlt_00388
-	-	SWS_Dlt_00389
-	-	SWS_Dlt_00390

-	-	SWS_Dlt_00392
-	-	SWS_Dlt_00393
-	-	SWS_Dlt_00394
-	-	SWS_Dlt_00395
-	-	SWS_Dlt_00396
-	-	SWS_Dlt_00397
-	-	SWS_Dlt_00399
-	-	SWS_Dlt_00402
-	-	SWS_Dlt_00403
-	-	SWS_Dlt_00404
-	-	SWS_Dlt_00405
-	-	SWS_Dlt_00406
-	-	SWS_Dlt_00407
-	-	SWS_Dlt_00408
-	-	SWS_Dlt_00410
-	-	SWS_Dlt_00411
-	-	SWS_Dlt_00412
-	-	SWS_Dlt_00414
-	-	SWS_Dlt_00415
-	-	SWS_Dlt_00416
-	-	SWS_Dlt_00417
-	-	SWS_Dlt_00422
-	-	SWS_Dlt_00423
-	-	SWS_Dlt_00424
-	-	SWS_Dlt_00425
-	-	SWS_Dlt_00427
-	-	SWS_Dlt_00428
-	-	SWS_Dlt_00449
-	-	SWS_Dlt_00450
-	-	SWS_Dlt_00451
-	-	SWS_Dlt_00469
-	-	SWS_Dlt_00471
-	-	SWS_Dlt_00472
-	-	SWS_Dlt_00474
-	-	SWS_Dlt_00475
-	-	SWS_Dlt_00480
-	-	SWS_Dlt_00481
-	-	SWS_Dlt_00482
-	-	SWS_Dlt_00483

-	-	SWS_Dlt_00484
-	-	SWS_Dlt_00485
-	-	SWS_Dlt_00487
-	-	SWS_Dlt_00488
-	-	SWS_Dlt_00489
-	-	SWS_Dlt_00490
-	-	SWS_Dlt_00491
-	-	SWS_Dlt_00492
-	-	SWS_Dlt_00493
-	-	SWS_Dlt_00494
-	-	SWS_Dlt_00495
-	-	SWS_Dlt_00496
-	-	SWS_Dlt_00497
-	-	SWS_Dlt_00498
-	-	SWS_Dlt_00499
-	-	SWS_Dlt_00501
-	-	SWS_Dlt_00502
-	-	SWS_Dlt_00503
-	-	SWS_Dlt_00504
-	-	SWS_Dlt_00505
-	-	SWS_Dlt_00506
-	-	SWS_Dlt_00507
-	-	SWS_Dlt_00508
-	-	SWS_Dlt_00509
-	-	SWS_Dlt_00513
-	-	SWS_Dlt_00514
BSW00431	-	SWS_Dlt_00511
BSW00434	-	SWS_Dlt_00511
SRS_BSW_00005	Modules of the æC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_Dlt_00511
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_Dlt_00239
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_Dlt_00511
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_Dlt_00511

SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_Dlt_00511
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_Dlt_00511
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Dlt_00511
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_Dlt_00511
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_Dlt_00468
SRS_BSW_00307	Global variables naming convention	SWS_Dlt_00511
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_Dlt_00511
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_Dlt_00511
SRS_BSW_00326	-	SWS_Dlt_00511
SRS_BSW_00327	Error values naming convention	SWS_Dlt_00447
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_Dlt_00511
SRS_BSW_00337	Classification of development errors	SWS_Dlt_00447
SRS_BSW_00339	Reporting of production relevant error status	SWS_Dlt_00511
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_Dlt_00511
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_Dlt_00511
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_Dlt_00239
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall	SWS_Dlt_00511

	be in place	
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_Dlt_00511
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_Dlt_00511
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_Dlt_00239
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_Dlt_00511
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_Dlt_00511
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_Dlt_00511
SRS_BSW_00376	-	SWS_Dlt_00511
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_Dlt_00238
SRS_BSW_00385	List possible error notifications	SWS_Dlt_00447
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_Dlt_00511
SRS_BSW_00387	The Basic Software Module specifications shall specify how the callback function is to be implemented	SWS_Dlt_00511
SRS_BSW_00395	The Basic Software Module specifications shall list all configuration parameter dependencies	SWS_Dlt_00511
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	SWS_Dlt_00511
SRS_BSW_00402	Each module shall provide version information	SWS_Dlt_00271
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_Dlt_00239
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_Dlt_00239

SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Dlt_00239
SRS_BSW_00409	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	SWS_Dlt_00511
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_Dlt_00511
SRS_BSW_00414	The init function may have parameters	SWS_Dlt_00239, SWS_Dlt_00437
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_Dlt_00511
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_Dlt_00511
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_Dlt_00511
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_Dlt_00511
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_Dlt_00511
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_Dlt_00511
SRS_BSW_00439	Enable BSW modules to handle interrupts	SWS_Dlt_00511
SRS_Dlt_00001	The DLT shall transmit log and trace messages from several sources over a communication interface to a receiving external client.	SWS_Dlt_00007, SWS_Dlt_00040, SWS_Dlt_00333, SWS_Dlt_00339, SWS_Dlt_00461
SRS_Dlt_00002	All log and trace messages sent by an ECU shall have a standardized transmission format and a standardized storage format.	SWS_Dlt_00039, SWS_Dlt_00043, SWS_Dlt_00117, SWS_Dlt_00301, SWS_Dlt_00302, SWS_Dlt_00467
SRS_Dlt_00003	SWCs shall have the possibility to send log or trace messages to the DLT module.	SWS_Dlt_00007, SWS_Dlt_00241, SWS_Dlt_00243, SWS_Dlt_00333
SRS_Dlt_00004	The DLT shall provide the actual set of log levels and the trace status to a SWC.	SWS_Dlt_00012, SWS_Dlt_00252, SWS_Dlt_00254, SWS_Dlt_00330

SRS_Dlt_00005	For each SWC the interface to DLT shall be configured.	SWS_Dlt_00289, SWS_Dlt_00345, SWS_Dlt_00426
SRS_Dlt_00006	Trace events from errors generated by BSW and SWCs shall be forwarded to the DLT module	SWS_Dlt_00430, SWS_Dlt_00431, SWS_Dlt_00432
SRS_Dlt_00007	The DEM shall forward error events to the DLT module	SWS_Dlt_00470, SWS_Dlt_00476, SWS_Dlt_00477, SWS_Dlt_00478, SWS_Dlt_00479
SRS_Dlt_00008	RTE shall provide an interface for DLT to trace RTE/VFB calls.	SWS_Dlt_00025, SWS_Dlt_00284
SRS_Dlt_00009	The DLT shall implement an interface to trace the RTE/VFB.	SWS_Dlt_00276, SWS_Dlt_00277, SWS_Dlt_00285
SRS_Dlt_00013	The transmitted data shall be packetized.	SWS_Dlt_00116, SWS_Dlt_00301, SWS_Dlt_00302, SWS_Dlt_00314, SWS_Dlt_00315
SRS_Dlt_00014	The transport format shall be binary.	SWS_Dlt_00097, SWS_Dlt_00304, SWS_Dlt_00378
SRS_Dlt_00016	The format shall deal with Big and Little Endianess.	SWS_Dlt_00097, SWS_Dlt_00304, SWS_Dlt_00458
SRS_Dlt_00017	Each log and trace message shall contain a timestamp, which will be added to the message during reception of the message in the DLT module	SWS_Dlt_00102, SWS_Dlt_00112, SWS_Dlt_00323, SWS_Dlt_00458
SRS_Dlt_00018	A global message counter shall be implemented, to detect messages loss.	SWS_Dlt_00084, SWS_Dlt_00105, SWS_Dlt_00106, SWS_Dlt_00319, SWS_Dlt_00458
SRS_Dlt_00019	For each log message, a log level shall be provided.	SWS_Dlt_00086, SWS_Dlt_00088, SWS_Dlt_00122, SWS_Dlt_00457
SRS_Dlt_00020	The log and trace message shall contain a parameter, which represents the source of the log and trace message	SWS_Dlt_00101, SWS_Dlt_00110, SWS_Dlt_00322, SWS_Dlt_00457
SRS_Dlt_00021	There shall be a logical grouping for log messages by using different identifiers.	SWS_Dlt_00127, SWS_Dlt_00128, SWS_Dlt_00457
SRS_Dlt_00022	Each ECU shall have its unique ECU ID	SWS_Dlt_00098, SWS_Dlt_00458
SRS_Dlt_00023	The payload shall transport the parameters of a log and trace message.	SWS_Dlt_00314, SWS_Dlt_00315, SWS_Dlt_00378, SWS_Dlt_00409, SWS_Dlt_00459
SRS_Dlt_00024	It shall be possible to transmit the parameters in a raw format.	SWS_Dlt_00096, SWS_Dlt_00310, SWS_Dlt_00418, SWS_Dlt_00460
SRS_Dlt_00025	It shall be possible to transmit ASCII text in log or trace messages.	SWS_Dlt_00352, SWS_Dlt_00400, SWS_Dlt_00420
SRS_Dlt_00026	The data in non-verbose mode shall be described by an extra file	SWS_Dlt_00398, SWS_Dlt_00401, SWS_Dlt_00418

SRS_Dlt_00027	Each message shall have a unique identifier significant for identifying the source of the tracing.	SWS_Dlt_00352, SWS_Dlt_00398, SWS_Dlt_00460
SRS_Dlt_00028	A control message shall be implemented to permit the external client to evaluate the round trip time.	SWS_Dlt_00207, SWS_Dlt_00208, SWS_Dlt_00221
SRS_Dlt_00029	A protection against unauthorized access in production phase shall be provided.	SWS_Dlt_00044, SWS_Dlt_00046, SWS_Dlt_00048, SWS_Dlt_00290
SRS_Dlt_00030	Monitoring and shaping of DLT log and trace event amount	SWS_Dlt_00054, SWS_Dlt_00055, SWS_Dlt_00056, SWS_Dlt_00202, SWS_Dlt_00344
SRS_Dlt_00031	The DLT shall be configurable at runtime.	SWS_Dlt_00068, SWS_Dlt_00069, SWS_Dlt_00071, SWS_Dlt_00077, SWS_Dlt_00079
SRS_Dlt_00032	A protocol shall be implemented to be able to set and query the trace status and log levels of log and trace sources of each ECU.	SWS_Dlt_00187, SWS_Dlt_00194, SWS_Dlt_00195, SWS_Dlt_00196, SWS_Dlt_00197, SWS_Dlt_00198, SWS_Dlt_00380, SWS_Dlt_00381, SWS_Dlt_00383
SRS_Dlt_00033	A list of all log and trace sources of an ECU shall be accessible from the external client.	SWS_Dlt_00012, SWS_Dlt_00021, SWS_Dlt_00059, SWS_Dlt_00064, SWS_Dlt_00197, SWS_Dlt_00245
SRS_Dlt_00034	DLT shall support a generic API for communicating over a DLT communication module.	SWS_Dlt_00040, SWS_Dlt_00042, SWS_Dlt_00043, SWS_Dlt_00263, SWS_Dlt_00264, SWS_Dlt_00265, SWS_Dlt_00272, SWS_Dlt_00273, SWS_Dlt_00461, SWS_Dlt_00462, SWS_Dlt_00463, SWS_Dlt_00515, SWS_Dlt_00516, SWS_Dlt_00517
SRS_Dlt_00035	The DCM shall provide an interface for DLT to transport log and trace messages over a diagnostic session.	SWS_Dlt_00037, SWS_Dlt_00039, SWS_Dlt_00339, SWS_Dlt_00340, SWS_Dlt_00434, SWS_Dlt_00435
SRS_Dlt_00036	The DLT shall provide a buffer for storing log and trace messages before initialization	SWS_Dlt_00003, SWS_Dlt_00004
SRS_Dlt_00037	There shall be a buffer to store log and trace message locally.	SWS_Dlt_00052, SWS_Dlt_00341, SWS_Dlt_00342
SRS_Dlt_00038	A mechanism shall be implemented to be able to set the trace status and log levels of registered application IDs and context IDs of each SWC.	SWS_Dlt_00012, SWS_Dlt_00058, SWS_Dlt_00059, SWS_Dlt_00252, SWS_Dlt_00254, SWS_Dlt_00330
SRS_Dlt_00039	The DLT shall provide the possibility to store configuration data in a persistent way.	SWS_Dlt_00073, SWS_Dlt_00074, SWS_Dlt_00076, SWS_Dlt_00077, SWS_Dlt_00287, SWS_Dlt_00288, SWS_Dlt_00452, SWS_Dlt_00453
SRS_Dlt_00040	the DLT component shall be able to filter log and trace	SWS_Dlt_00012, SWS_Dlt_00065, SWS_Dlt_00067, SWS_Dlt_00068,

	messages	SWS_Dlt_00071, SWS_Dlt_00330, SWS_Dlt_00347
SRS_Dlt_00041	DLT shall be a central software component in BSW for the log and trace functionality.	SWS_Dlt_00464
SRS_Dlt_00042	The Log and trace SW component shall be part of the system during production phase	SWS_Dlt_00465, SWS_Dlt_00466
SRS_Dlt_00044	There shall be the possibility to transmit the parameters with additional information about themselves (self-description).	SWS_Dlt_00119, SWS_Dlt_00135, SWS_Dlt_00303, SWS_Dlt_00421, SWS_Dlt_00459

7 Functional specification

[SWS_Dlt_00464] ⌈Dlt (**Diagnostic Log and Trace**) is a basic software module, which handles and stores log and trace messages produced by SW-C it self or the interactions between SW-C and RTE/VFB and by the Basic Software Modules Dem and Det. The log and trace messages are generated by calling APIs provided by the Dlt module.⌋ (SRS_Dlt_00041)

[SWS_Dlt_00466] ⌈Dlt shall be available in development and in production phase.⌋ (SRS_Dlt_00042)

7.1 Dlt term definition

This chapter describes the parameters and content of a log and trace message and additional terms.

7.1.1 Log and trace message

A log and trace message contains all data and options to specify a log and trace event in a software. The log and trace message internally consist of a header and payload.

7.1.2 User

The user of Dlt is the programmer of the software, which uses the Dlt API to generate log and trace messages.

7.1.3 Log

The user generates log messages on demand. Each time the user wants to show some information about state changes or value changes, he adds an API call to Dlt.

7.1.4 Trace

Trace messages can be generated by instrumentation of the code (e.g. VFB traces). The instrumented code calls the API of Dlt.

7.1.5 ECU ID

ECU ID is the name of each ECU composed by four 8 bit ASCII characters for example ABS0 or COMB. If not all four characters are used the remaining characters shall be filled by null.

7.1.6 Session ID

Session ID is the identification number of a log or trace session. A session is the logical entity of the source of log or trace messages. If a SW-C is instantiated several times a new session with a new Session ID for every instance is used. A SW-C additionally can have several log or trace sessions if it has several ports opened to Dlt.
Since Session ID is not specified in AUTOSAR for SW-C the port defined argument values (see chapter 7.6.4) shall be used for this number. For BSW modules Dem and Det it is the module Id.

7.1.7 Application ID

Application ID is a abbreviation of the SW-C/BSW modules Dem and Det. It identifies the SW-C/BSW module in the log and trace message. It is composed by four 8 bit ASCII characters for example Det, Dem or ABS. If not all four characters are used the remaining characters shall be filled by null.

7.1.8 Context ID

Context ID is a user defined ID to group log and trace messages produced by a SW-C/BSW modules Dem and Det to distinguish functionality. Each Application ID can own several Context IDs. Context ID's are grouped by Application ID's. Context IDs shall be unique within an Application ID.
The identification of the source of a log and trace message is done with a pair of Application ID and Context ID.
It is composed by four 8 bit ASCII characters. If not all four characters are used the remaining characters shall be filled by null.

7.1.9 Message ID

Messaged ID is the ID to characterize the information, which is transported by the message itself. A Message ID identifies a log or trace message uniquely. It can be used for identifying the source (in source code) of a message and it can be used for characterizing the payload of a message. A message ID is statically fixed at development or configuration time.

7.1.10 Log level and trace status

A log level defines a classification for the severity grade of a log message.

The trace status provides information if a trace message should be send.

7.1.11 Time

Each log and trace message may contain a time attribute. The time attribute is a free defined time-value. It is the time since the start of the ECU. (see 7.6.8.1 and 7.7.3.6)

7.1.12 Payload

The Payload contains the message ID and user defined information of a log and trace message.

7.1.13 External client

An external client is a tool, which can be run on a PC or another ECU, which is connected to Dlt over Dcm [6] or over the Dlt communication module.

7.2 Use Cases for logging and tracing with Dlt

7.2.1 Use Case general logging with Dlt

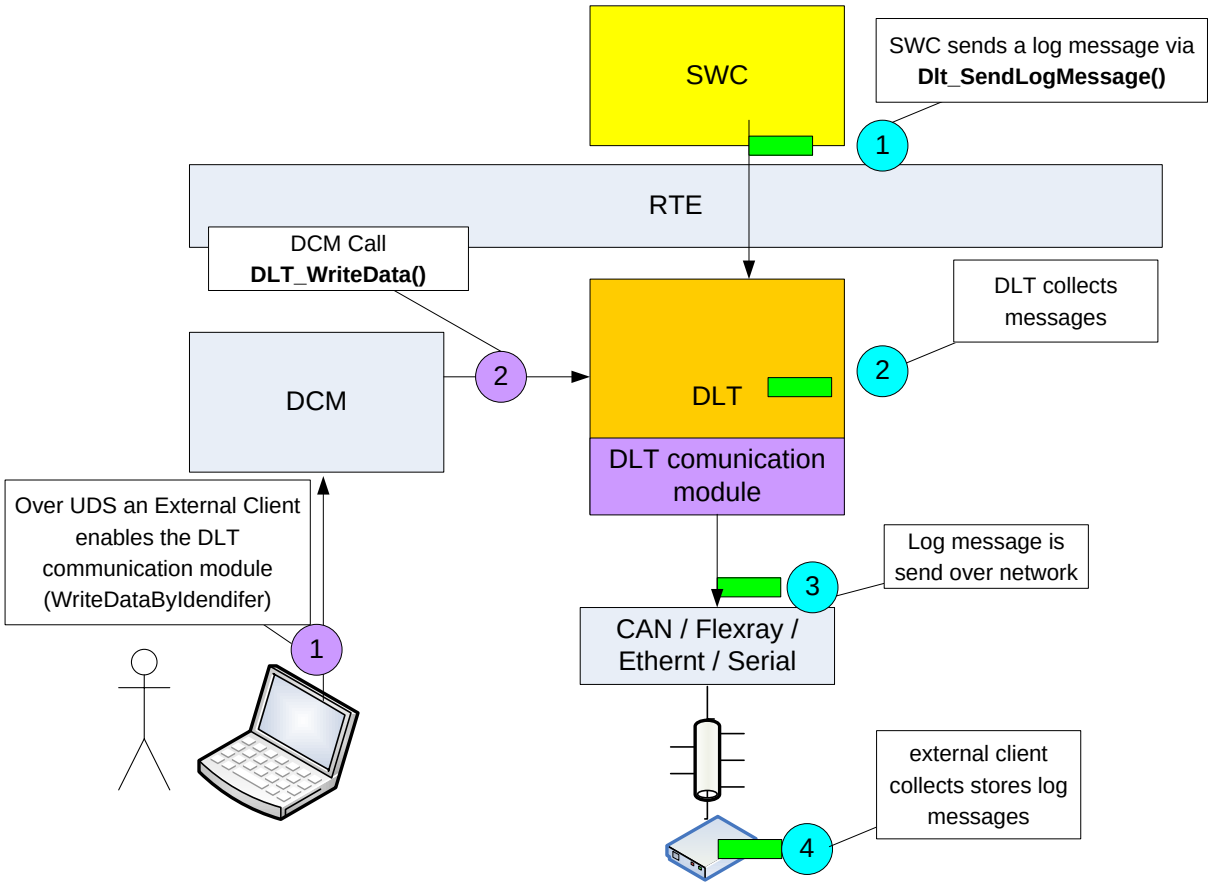


Figure 5: Use Case general logging with Dlt

The Dlt communication module is enabled by an external client. The external client has to set up a diagnostic session in a defined security level and send control message to Dlt for enabling the Dlt communication module. A SW-C is generating a log message. The log message is sent to Dlt by calling the API provided by Dlt. Dlt sends the log message to the implemented Dlt communication module interface.

7.2.2 Use Case logging over UDS with Dlt

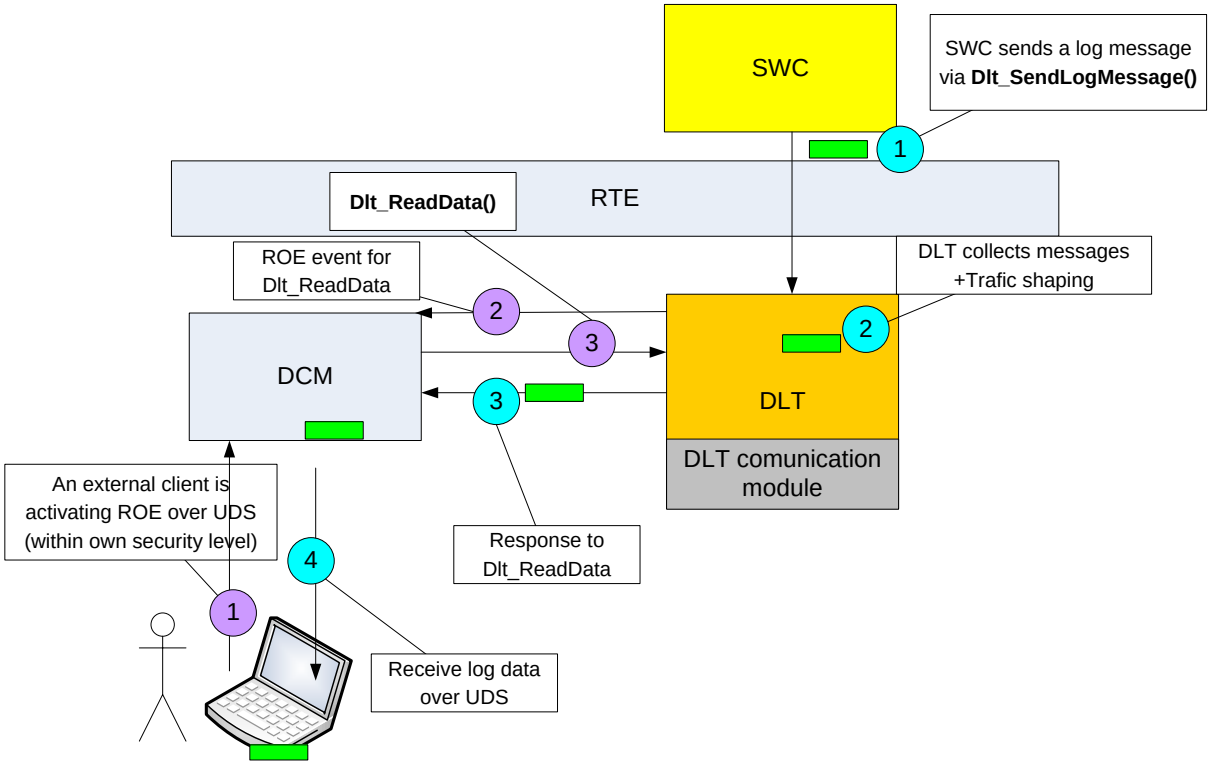


Figure 6: Use Case logging over UDS with Dlt

An external client enables the communication by setting up a diagnostic session in a defined security level. A SW-C is generating a log message. The log message is sent to Dlt by calling the API provided by Dlt. Dlt sends the log message over Dcm to the external (diagnostic) client.

7.2.3 Use Case tracing of VFB

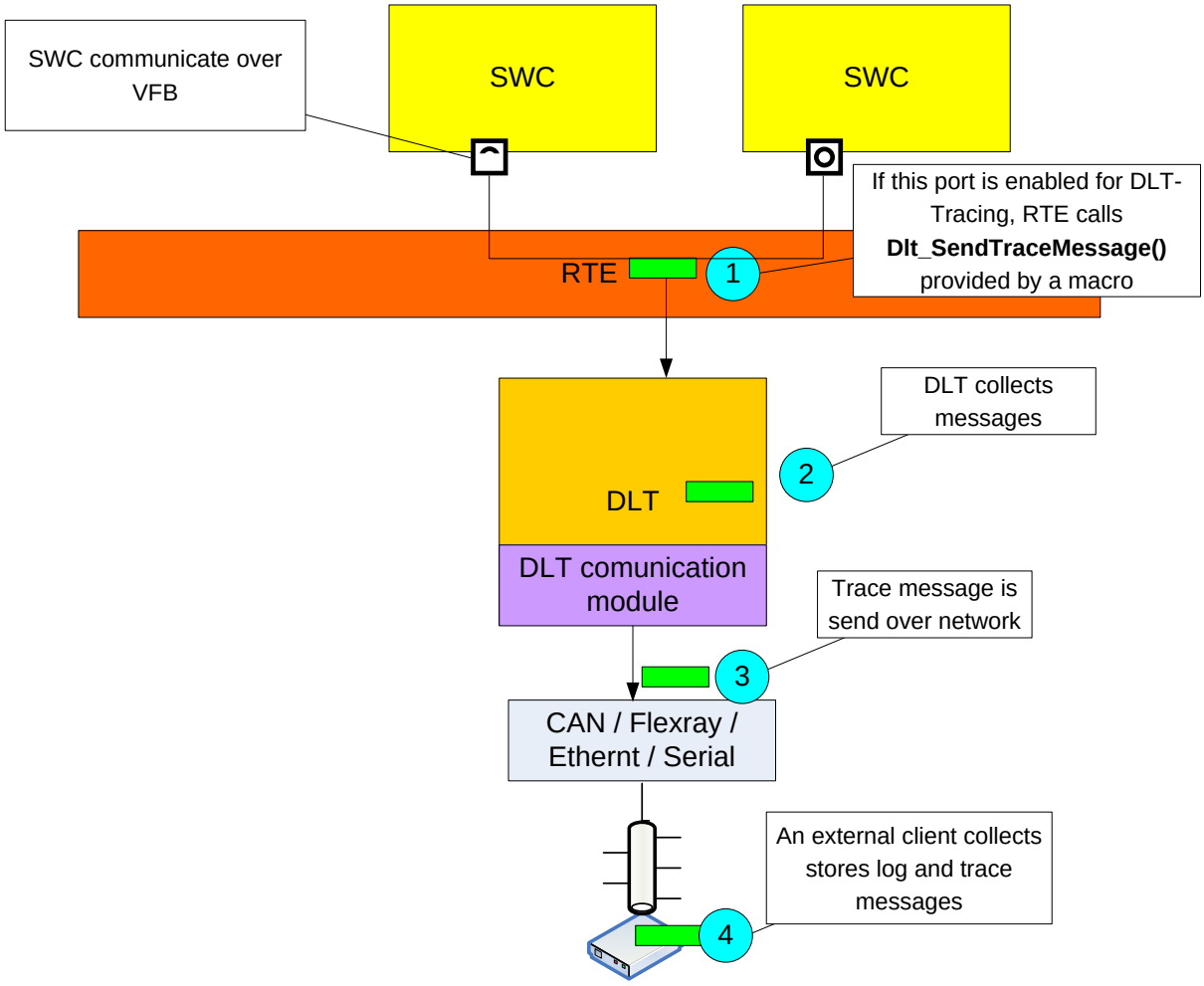


Figure 7: Use Case tracing of VFB

RTE calls the macro provided by Dlt, which calls the Dlt API generating the trace message. Dlt sends the trace message to the implemented Dlt communication module interface. The Dlt communication module forwards the trace message to the network. An external client receives and stores the trace messages from Dlt.

7.2.4 Use Case runtime configuration of Dlt

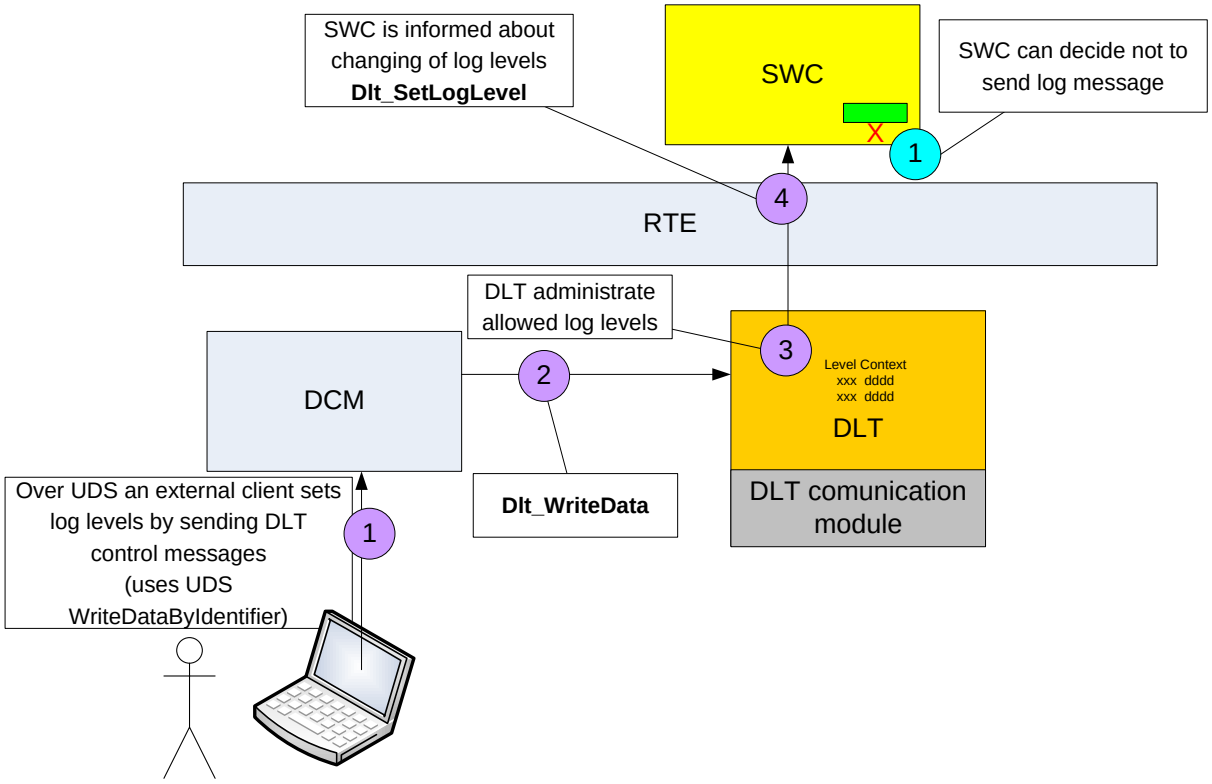


Figure 8: Use Case runtime configuration of Dlt

An external (diagnostic) client enables the communication by setting up a diagnostic session in a defined security level. The external client sets the log and trace level in Dlt. Dlt invokes the client-server interface of the SW-C to inform the SW-C about the new log level.

7.2.5 Use Case Dlt interaction only over Dlt communication module

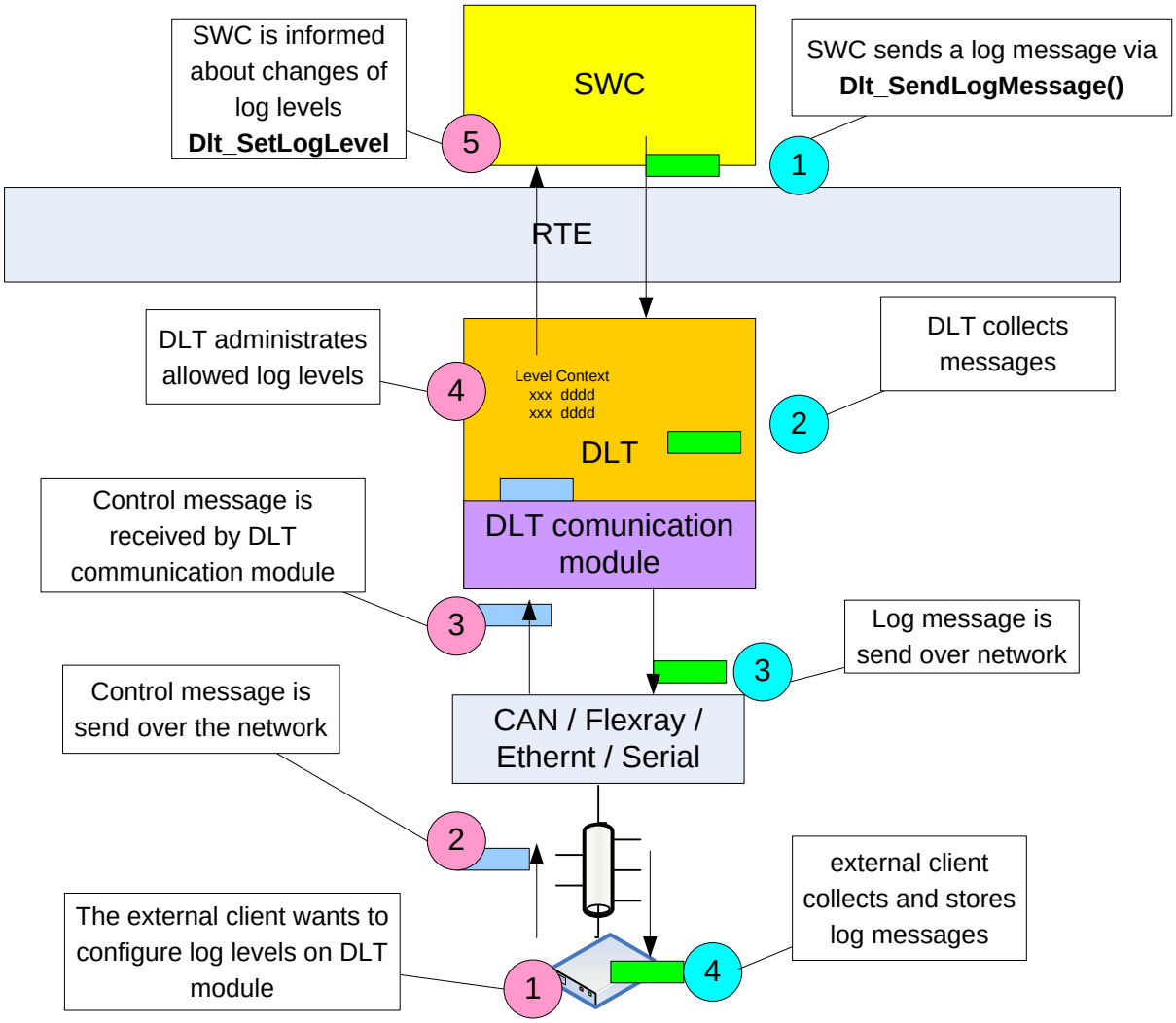


Figure 9: Communication of an external client only over the Dlt communication module

En external client is connected over a standard network/interface of the ECU to the Dlt. In this case not the UDS interface over the Dcm is used and a high bandwidth interface of the ECU can be used. The external client sends some control messages to the Dlt module to configure some log levels. In the other direction the Dlt sends log or trace messages to the external client.

7.3 Internal behavior of Dlt Module

7.3.1 Overview

The Dlt module communicates with the software modules, which generate the log and trace messages, and with the external client used by an operator. Different phases of a software life cycle (developing and production) shall be considered.

The Dlt module shall support the various functionality in the different operating modes of the ECU (start-up, runtime, shutdown).
The following figure shows a very high-level architecture for the component. This architecture is only a logical representation.

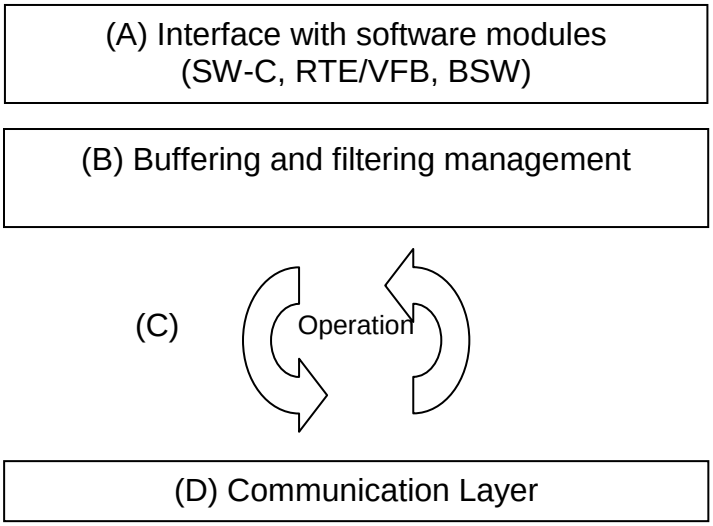


Figure 10 Overview Dlt module logical representation

(A) provides the interface used by the software module (SW-Cs, BSW modules) to generate the log and trace message.
(B) manages the buffering of the messages received by SW-Cs or BSW modules before they are sent over the network to the external client. Also the implementation of additional filtering is done by this module.
(C) is the core of Dlt which has to communicate with the external client to engage the necessary actions for driving the behavior of Dlt itself. This component has multiple objectives:

1. interpret and implement the commands from the external client
2. extracting the messages from the buffer to be sent to the external client
3. manage the application protocol to transport the log and trace message to the external client.

(D) represents the functionality of Dlt for communication with the external client.

7.3.1.1 Buffer strategy

The Dlt module shall implement a buffer management to solve the following issues:

- the rate of receiving log and trace messages by the SW-Cs is temporary faster than the communication channels available bit rate
- to store some log and trace messages if no external client is connected

The overall definition of the characteristic of the buffer and its management policy are up to the implementation.

[SWS_Dlt_00490] ⌈Dlt control messages (see 7.7.7.1) should be handled separately. If any control messages are to send, this messages should be send before any normal log/trace message.⌋ ()

7.3.1.2 Runtime configuration

Dlt provides the functionality for an external client to change the log level or trace status of the registered Context IDs and Application IDs. It shall be secured by running a diagnostic session.

As explained in chapter 7.3.3.1.6 each SW-C has to register it's Context ID's and Application IDs. Dlt sets up a table of all registered pairs of Application IDs and Context ID. At every message reception, the Dlt module shall check the log level of the provided messages (see 7.6.5). If the log level of the message is in the pass through range, the message will be forwarded to the external client.

Example of pass through range:
The pass through range is the range of log levels of log messages to forward to an external client. For example the pass through range is set to log level 0 to log level 4, all messages with in log level 0-4 are passed all other (5-7) are rejected.

Runtime configuration of Dlt means, that the pass through range shall be modifiable at runtime.

7.3.2 Startup and Shutdown behavior

Dlt is using the NVRamManager and is to be initialized very late in the ECU startup phase. The Dlt_Init() function should be called after the NVRamManager is initialized.

Because of BSW modules Dem and Det may send log messages to Dlt before Dlt is initialized, Dlt shall handle this data in accurate way.

[SWS_Dlt_00003] ⌈Dlt shall have a temporary C-initialized buffer (init buffer), where only the provided data from the Dlt_SendLogMessage are stored. The size of this buffer is configurable with DltInitBufferSize.⌋ (SRS_Dlt_00036)

NOTE: After initialization, it is possible to re-use the buffer within the normal message buffer or to use the normal message buffer as temporary buffer before initialization.

[SWS_Dlt_00004] ⌈Only if Dlt is not initialized, Dlt shall store these data in the init buffer. When the init-routine is called this init buffer shall be read and the stored data encapsulated in a complete log and trace message shall be forwarded to the Dlt message buffer.⌋ (SRS_Dlt_00036)

[SWS_Dlt_00005] ⌈If Dlt module is initialized, the incoming data from BSW modules shall be directly encapsulated in a log and trace message and forwarded to the Dlt message buffer.

Because at ECU startup before Dlt initialization a correct time is not available the following behavior should be implemented.⌋ ()

[SWS_Dlt_00483] ⌈The timestamp field (TMSP) of the transferred log or trace message (see 7.7.3.6) for all log or trace messages transferred from the init buffer to the Dlt (see **[SWS_Dlt_00004**) shall be zero (0x0).⌋ ()

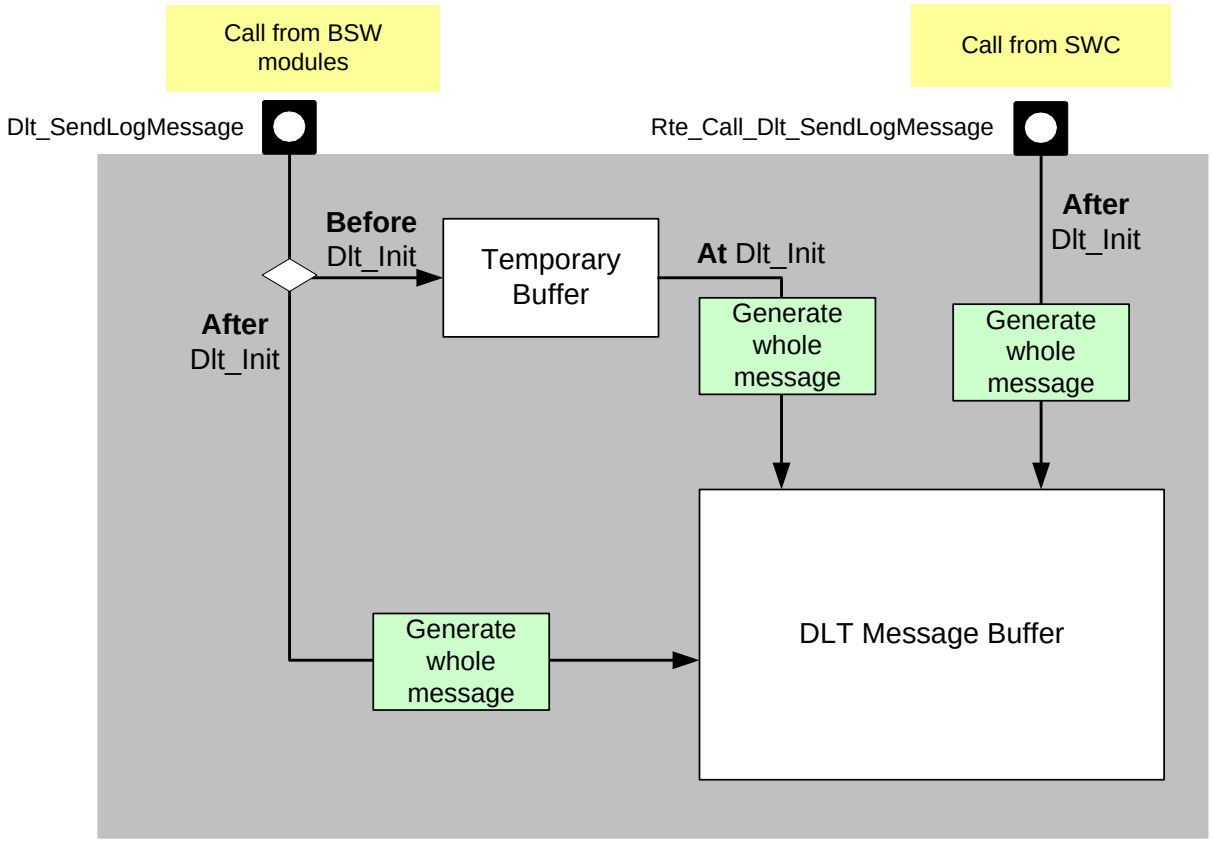


Figure 11 For system startup Dlt shall have a temporary buffer where incoming messages from BSW modules can be stored.

7.3.3 Communication with producer of log and trace messages

There are two kinds of communication with the Dlt module. The connection between the ECU and the external client can be a communication interface like Ethernet or a serial line, a standard CAN or FlexRay interface with reserved messages for Dlt communication or at least a connection over the OBD-Diagnostic connector of the vehicle with the UDS protocol.
There are different communication scenarios. On one side the Dlt module collects log and trace messages from the SW-Cs on the other side Dlt shall transport the log and trace messages to an external client, which can store these data permanently.
Additionally for changing the log levels at runtime, Dlt shall receive commands from an external client.

7.3.3.1 Communication with SW-C

The goal of Dlt is to collect log and trace messages. In the view of a SW-C, it shall provide an interface for SW-Cs for explicitly sending log or trace messages.
This interface shall be a ClientServerInterface, where the Dlt module provides the port (it is the server). All SW-Cs can call the DltService port to forward log or trace messages to the Dlt module.

[SWS_Dlt_00007] ⌈The function `Dlt_SendLogMessage()` shall be called for providing a log message.⌋ (SRS_Dlt_00001, SRS_Dlt_00003)

[SWS_Dlt_00333] ⌈The `Dlt_SendTraceMessage()` shall be called for providing a trace message by a SW-C.

Every call from the SW-C to Dlt contains an assigned Application ID and Context ID. This pair of IDs ensures the correct reassignment to a specific part of an application.
Dlt itself assigns to every pair a log level or a trace status for filtering the messages at receiving time.

Each SW-C shall register all pairs of Application- and Context IDs to Dlt. Because Dlt shall handle specific log levels and trace status for several Application IDs and Context IDs (see 7.3.3.1.2 and 7.3.3.1.6) it is important for Dlt to know the corresponding Application and Context IDs for each connected port interface.⌋ (SRS_Dlt_00001, SRS_Dlt_00003)

7.3.3.1.1 Sending messages to Dlt

[SWS_Dlt_00009] ⌈For sending log or trace messages from a SW-C to Dlt every time an Application ID and Context ID shall be assigned to the message.⌋ ()

[SWS_Dlt_00295] ⌈For reducing the overall system load, SW-Cs shall check (before sending a log message to Dlt), that the log level of the Context ID is the same or a higher log level as in the message.

The `Dlt_SendLogMessage()` interface is the logging interface for SW-Cs. Here a software developer (user of Dlt) can explicitly provide some information for logging the SW-C's behavior.⌋ ()

[SWS_Dlt_00010] ⌈Within a log message, a log level shall be provided. Possible values of log levels are:

DLT_LOG_FATAL	fatal system errors, should be very rare
DLT_LOG_ERROR	errors occurring in a SW-C with impact to correct functionality
DLT_LOG_WARN	log messages where a incorrect behavior can not be ensured

DLT_LOG_INFO	log messages providing information for better understanding the internal behavior of a software
DLT_LOG_DEBUG	should be used for messages which are only for debugging of a software usable
DLT_LOG_VERBOSE	log messages with the highest communicative level, here all possible states, information and everything else can be logged

Table 7-1 Log levels defined, most important is DLT_LOG_FATAL and less important is DLT_LOG_VERBOSE

⌋()

[SWS_Dlt_00011] ⌈The Dlt_SendTraceMessage() function may be in production phase a dummy function.

Trace messages can be something like a trace of starting and returning a function or tracing variables at some points and so on. In a SW-C the code can be instrumented automatically by some tools for providing a trace in significant manner. These tools are used for generating e.g. code coverage or function coverage and so on.⌋()

[SWS_Dlt_00296] ⌈For these purposes, the following trace events shall be used:

DLT_TRACE_VARIABLE	for tracing the value of a variable
DLT_TRACE_FUNCTION_IN	for tracing the calling of a function
DLT_TRACE_FUNCTION_OUT	for tracing the returning of a function
DLT_TRACE_STATE	for tracing a state of a state machine

Table 7-2 Trace info for providing a trace message

⌋()

7.3.3.1.2 Notifying SW-C about change of log level or trace status of a Context ID

The log level or trace status of a Context ID can change at runtime.

[SWS_Dlt_00012] ⌈To be notified of a changing event a SW-C shall provide the system service interface `LogTraceSessionControl`.

The Dlt module shall know the relation between the Session ID (the port defined argument value) and the specific `LogTraceSessionControl` interface of a SW-C.

This relation is specified at configuration time and can be extracted from the specific port interface of a SW-C/runnable (see chapter 7.6.4.1.).

Each SW-C shall provide such an interface if it uses the Dlt service. It shall store the state of the log level and trace status locally. Before a call to `Dlt_SendLogMessage()or Dlt_SendTraceMessage()`is done the stored log level shall be checked. The call shall only be done if the log level of the message to be sent is the same or a more important log level. This is for reducing consumption of ECU performance. (see Figure 12)⌋ (SRS_Dlt_00004, SRS_Dlt_00033, SRS_Dlt_00038, SRS_Dlt_00040)

[SWS_Dlt_00330] ⌈If a log level or trace status of a specific pair of Application ID and Context ID changes, Dlt shall notify the registered SWS/runnable through out the corresponding interface.⌋ (SRS_Dlt_00004, SRS_Dlt_00038, SRS_Dlt_00040)

[SWS_Dlt_00331] ⌈The function to call shall be taken from the table specified in chapter 7.6.4.1.

This function is not a direct call to the SW-C's interface. It is a call to the RTE with the RTE API Rte_call_XXX (the correct linker symbol is something like Rte_call_XXX). The RTE forwards the call to the SW-C/runnable.

For a description of managing pairs of Application ID and Context ID and its corresponding log levels see chapter 7.6.4.

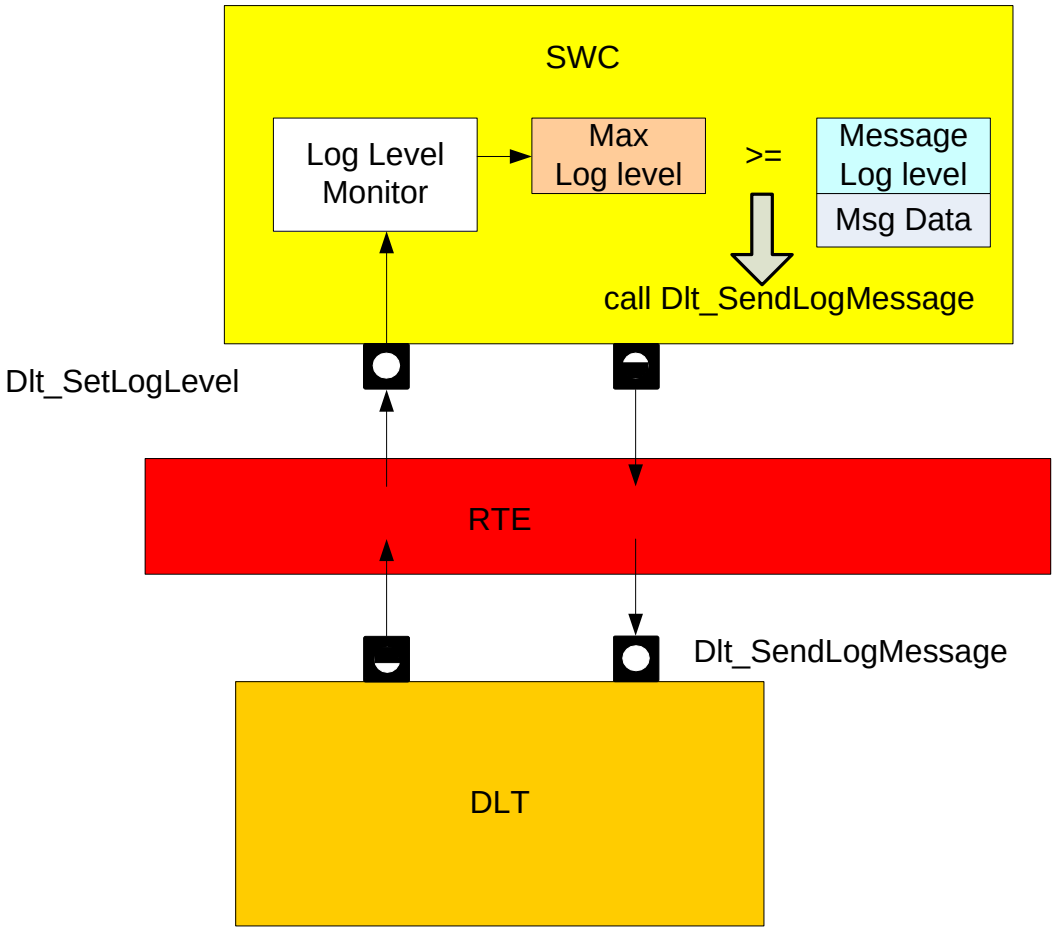


Figure 12 SW-C shall be aware of the status of log level and trace status

⌋()

7.3.3.1.3 Message Filtering by SW-C

All SW-Cs shall check their log level or trace status of a Context ID and Application ID pair. A SW-C/runnable shall hold the maximum allowed log level and the enable flag for tracing, for a given pair of Application ID and Context ID. Before a message is sent to Dlt, these values shall be checked and only if the log level of the message is in the pass-through range, the message shall be send.

```
if (message_log_level <= max_log_level){
     Dlt_SendLogMessage(…);
}
```

Figure 13 Code sample for SW-C to check the log level of the message before call of Dlt_SendLogMessage

If a SW-C doesn't check the log level of a log message, the log message is send to Dlt. Then if the feature DltImplementFilterMessages is enabled, the message is filtered by the Dlt again. If this feature is not enabled, the message is transmitted to the external client (if connected).
So it is up to the developer to implement this feature. If he do not want to filter messages in the SW-C he can enable the feature DltImplementFilterMessages and the Dlt will do the filtering. But this increases ECU load.

7.3.3.1.4 Notifying SW-C about switch between Verbose Mode and Non Verbose Mode

If on an ECU the Verbose Mode is supported, the SW-C shall be informed when Dlt switches between Non Verbose Mode and Verbose Mode. Because of the payload provided by a call to Dlt_SendXXXMessage is either in Verbose Mode or Non Verbose Mode.

[SWS_Dlt_00014] ⌈It shall be possible to enabled or disabled this feature via the configuration Parameter DltImplementVerboseMode.⌋()

[SWS_Dlt_00015] ⌈Dlt shall call the interface VerboseModeControl with the function Dlt_SetVerboseMode of each registered SW-C/runnable if the state of verbose mode changes.
The identification of the correct port to call by Dlt shall work in the same way as it works by calling Dlt_SetLogLevel (see 7.3.3.1.2, 7.3.3.1.7 and 7.6.4.1).⌋()

[SWS_Dlt_00016] ⌈In the case of implementing this feature, the tables specified in 7.6.4.1 shall be expanded with the pointer to the provided functions by the RTE. (The RTE forwards the call to the SW-Cs)⌋()

7.3.3.1.5 Injection of function calls in SW-C from Dlt

This functionality is for injection function calls in SW-Cs. This shall only be used for testing issues. The intention for this feature is an extended testing procedure where some special functions in the SW-C are provided for triggering some testing procedures.

[SWS_Dlt_00017] ⌈This feature shall be enabled and disabled via the configuration parameter DltImplementSWCInjection.⌋ ()

[SWS_Dlt_00018] ⌈Dlt calls the interface InjectionCallback of each registered SW-C/runnable. The identification of the correct port to call by Dlt shall work in the same way as it works by calling Dlt_SetLogLevell (see 7.3.3.1.2, 7.3.3.1.7 and 7.6.4.1).⌋ ()

[SWS_Dlt_00019] ⌈The function to call by Dlt is Dlt_InjectCall (the correct linker symbol is something like Rte_call_XXX). In the case of implementing this feature, the tables specified in 7.6.4.1 are to expand with the pointer to the provided functions by the RTE.⌋ ()

7.3.3.1.6 Registering Context IDs and Application IDs to Dlt

Dlt shall handle a log level and a trace state for every pair of Context ID and Application ID. To know what pairs are defined in an ECU a SW-C shall register this pairs at runtime to the Dlt module. Because of the developing of SW-C shall not be object of this specification the Dlt module shall collect this information at runtime. In addition a dynamic registration supports the possibility to see which SW-C/runnable is active and which not.
When a SW-C is calling the `Dlt_RegisterContext()` method of the DLTServicel interface, a port defined argument value is provided (SessionID) to the Dlt module.
The value of this port defined argument corresponds to LogTraceSessionControl interface of the SW-C/runnable for providing information about the changing of a log level to the SW-C/runnable.

[SWS_Dlt_00021] ⌈Dlt shall remember the relation between the registered Application ID + Context ID and the port interface where this pair is registered.⌋ (SRS_Dlt_00033)

7.3.3.1.7 Port Defined Argument Values and LogTraceSessionControl interface

For every function call of Dlt_SendLogMessage, Dlt_SendTraceMessage and Dlt_RegisterContext a Port Defined Argument Value shall be provided.

[SWS_Dlt_00022] ⌈This Port defined Argument Values shall be used by Dlt as session identifiers.

A session is the part of a SW-C for which a log level monitor (see Figure 12) is responsible. For each log level monitor the same session number (Port Defined Argument Value) shall be used.」()

[SWS_Dlt_00023] 「The port defined argument value corresponds to the defined Session ID (in this document). The value shall start at 0x1000 (for BSW modules the module ID is taken, starts at 0x0).」()

[SWS_Dlt_00332] 「For connecting a Dlt Service Port to a SW-C the port defined argument value is incremented every time. Therefore, the value is of 0x1000 + n, where n is a continuous number.」()

7.3.3.2 VFB-Trace

In contrast to the communication with SW-Cs the meaning of trace is different here. The VFB-Trace is specified in RTE [9] and VFB [10]. This chapter describes the interaction of the Dlt module with the VFB-Trace and the internal control of the trace data.
The meaning VFB-Trace is an implicit (system inherent) forwarding of SW-C communication data (which flows over the RTE) to the Dlt module. Trace means in this case that no explicit call by the SW-C is made to forward this data to Dlt.

[SWS_Dlt_00024] 「All explicit communication mechanism used by a SW-C shall be traceable by Dlt. These are data, transported over a Client-Server-Port or a Sender-Receiver-Port.」()

[SWS_Dlt_00334] 「In addition the implicit communication over a Sender-Receiver-Port shall be traceable.」()

7.3.3.2.1 Interfaces provided by Dlt for VFB-Trace

VFB-Trace needs from the Dlt module the header file Dlt_Rte_Hook.h. In this file Dlt tells the RTE which traces to enable. This works simply by providing some #defines in this header file. Each define stands for a separate hook function in the VFB-Trace. This hook function is than called by the RTE if the corresponding RTE-API is called. The hook function it self is provided by the Dlt module.
To ensure the correct prototype of this function the RTE module exports all expected prototypes in it BSW-ModulDescription.

[SWS_Dlt_00025] 「The VFB-Trace works with defines included by the RTE at building time. These defines shall be provided by Dlt in the file Dlt_Rte_Hook.h.」(SRS_Dlt_00008)

[SWS_Dlt_00284] 「Dlt shall be compliant to the VFB-Trace described in the AUTOSAR_RTE_SWS [9].」(SRS_Dlt_00008)

[SWS_Dlt_00276] ⌈Dlt shall provide the possibility to trace Client-Server communication as well as Sender-Receiver communication. The RTE – API functions Rte_Send, Rte_Write, Rte_Receive, Rte_Read, Rte_Call and Rte_Result shall be support at least.⌋ (SRS_Dlt_00009)

[SWS_Dlt_00026] ⌈Every signal or event which shall be traceable by Dlt shall be configured at configuration time. The so called event-name in VFB Trace or function name for each trace event is provided in the configuration container DltVfbTrace within the configuration parameter `DltVfbTraceFunction`.⌋ ()

[SWS_Dlt_00027] ⌈Dlt shall provide the implementation of the hook functions for every configured event given by DltVfbTraceFunction in the file Dlt_Rte_hook.c⌋ ()

[SWS_Dlt_00335] ⌈The prototype of this function is to take from the BSW-ModulDescription of the RTE.⌋ ()

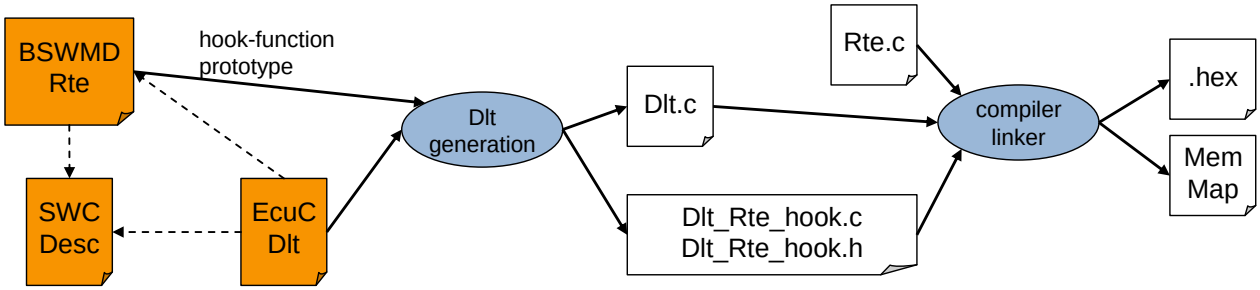


Figure 14 Dependencies and generation process for VFB-Trace implementation in Dlt.

7.3.3.2.2 Generating hook functions

[SWS_Dlt_00285] ⌈Because of the interface Dlt_SendTraceMessage is a SW-C interface, a internal function which is equivalent to Dlt_SendTraceMessage shall be implemented to call by the generated hook functions.⌋ (SRS_Dlt_00009)

[SWS_Dlt_00277] ⌈In the hook function the internal representation of Dlt_SendTraceMessage shall be called. This call shall be in non verbose mode.⌋ (SRS_Dlt_00009)

[SWS_Dlt_00278] ⌈The payload for this call shall be filled with the arguments provided by the hook function. All data transported with the arguments shall be provided.⌋ ()

[SWS_Dlt_00336] ⌈If pointers to structures are given, the structure shall be interpreted and send to the internal representation of Dlt_SendTraceMessage.

For more information about the contents of the arguments the information from the corresponding SW-C Description can be taken. This is to find with the event name over the ECU configuration.⌋()

[SWS_Dlt_00279] ⌈Every hook function shall get its own Context ID.
In some cases some events can be bundled to the same Context ID. This shall mostly be done if e very large number of signals are to trace.⌋()

[SWS_Dlt_00337] ⌈The Application ID shall be “VFBT” ⌋()

[SWS_Dlt_00484] ⌈Message Type (MSTP) entry in the generated trace message shall be DLT_TYPE_NW_TRACE, the Message Trace Info (MSTI) entry in this case shall be DLT_NW_TRACE_IPC.⌋()

[SWS_Dlt_00280] ⌈Because of non-verbose mode is used, a unique Message ID shall be generated for each call to Dlt_SendTraceMessage.⌋()

[SWS_Dlt_00338] ⌈Additionally the description (see 7.7.5.1.3) for this Message ID - payload shall be generated and provided.
This description can be generated from the SW-C description file, were the interface is described.⌋()

[SWS_Dlt_00281] ⌈In each hook function the trace status of the Context ID shall be checked.

```
if (vfb_actual_trace_status_contextXY){
     Dlt_SendTraceMessage(…);
}
```

Figure 15 Requirement for hook function to check the trace status of the Context ID before call of Dlt_SendTraceMessage (`vfb_actual_trace_status_contextXY` is a freely named variable to hold the actual trace status for a specific Context ID)

Dlt shall handle for every VFB-Trace hook function an own Context ID and for that reason Dlt shall handle for every VFB-Trace Context ID a separate trace status. This can be done with a separate variable (compare vfb_actual_trace_status_contextXY in Figure 15) ⌋()

[SWS_Dlt_00283] ⌈A separate function shall be implemented to modify the trace status of VFB-Trace hook functions. This function shall be harmonized with the SW-C LogTraceSessionControl interfaces (see chapter 7.6.4) ⌋()

7.3.3.3 Log Messages from BSW-Modules Dem and Det

For a better understanding of system behavior in the case of a system error, the BSW modules Dem and Det shall forward incoming reported events to the Dlt module.
The Dem in this case shall notify Dlt if the status of events changes. The Det shall forward reported development errors to the Dlt module.

7.3.3.3.1 Log Messages from Dem

Dem [5] stores internally events generated by SW-Cs and BSW modules. These events are characterized by event IDs.
To an event in Dem belonging some additional information. This are a Diagnostic Trouble Code (DTC), Extended Data Records and a Freeze Frame. Each time the state of an event changes Dem calls the Dlt_DemTriggerOnEventStatus() function to notify Dlt of this change.
Always use "DEM_DTC_FORMAT_UDS" as parameter for getting the DTCCode.

[SWS_Dlt_00474] ⌈Dlt shall provide the function Dlt_DemTriggerOnEventStatus.

Dem provides within this function the EventID of the event which status has changed. With this EventId Dlt shall request additional information about the event.⌋()

[SWS_Dlt_00475] ⌈In Dlt_DemTriggerOnEventStatus Dlt shall compare the EventStatusOld and EventStatusNew. If the event status is not the same Dlt shall build a Dlt log-message with the status of this event and send it by calling internally Dlt_SendLogMessage().⌋()

[SWS_Dlt_00476] ⌈The log message generated for Dem Events shall have the following payload entries:

Number	***Type***	***Name***	***Description***
1	uint32	EventId	the EventId
2	uint32	DTCOfEvent	the DTC of the Event
3	RAW	EventExtendedDataRecord	all extended data records
4	RAW	EventFreezeFrameData	the most resent FreezeFrame

Table 7-3 The payload attached the log message generated for an event change by Dem (See 7.7.5 Payload and 8.4.2.1 Dlt_SendLogMessage)

⌋(SRS_Dlt_00007)

[SWS_Dlt_00377] ⌈The ApplicationID, ContextID and MessageID of the send log message shall have the following values:

ApplicationID = “DEM”
ContextID = “STD0”

MessageID = 0x00000001

⌋()

[SWS_Dlt_00477] ⌈The DTCOfEvent entry from [SWS_Dlt_00476] shall be requested from the Dem by calling the function Dem_GetDTCOfEvent() with the EventId provided in Dlt_DemTriggerOnEventStatus() and DTCFormatType to the appropriate format of the DTC value.⌋(SRS_Dlt_00007)

[SWS_Dlt_00478] ⌈The EventExtendedDataRecord entry from **[SWS_Dlt_00476]** shall be filled by calling the Dem_DltGetAllExtendedDataRecords() function of Dem with the EventId provided in Dlt_DemTriggerOnEventStatus().⌋(SRS_Dlt_00007)

[SWS_Dlt_00479] ⌈The EventFreezeFrameData entry from **[SWS_Dlt_00476]** shall be filled by calling the Dem_DltGetMostRecentFreezeFrameRecordData() function of Dem with the EventId provided in Dlt_DemTriggerOnEventStatus().⌋(SRS_Dlt_00007)

NOTE: The data in the ExtendedDataRecord and the FreezeFrame are not interpreted by the Dlt module. They are send as raw data and the interpretation should be done at the external client. There some description files like specified in the ODX standard [iii] could be used.

7.3.3.3.2 Log Messages from Det

SW-Cs and BSW modules report errors to the Det module [11]. Such errors shall be forwarded to Dlt as messages with a suitable content using the function Dlt_DetForwardErrorTrace ().
All parameters from the Det function Det_ReportError() shall be forwarded to Dlt function Dlt_DetForwardErrorTrace () by the Det fan-out capability.

[SWS_Dlt_00430] ⌈Dlt shall provide the Dlt_DetForwardErrorTrace () function for the fan-out capability of Det.⌋(SRS_Dlt_00006)

[SWS_Dlt_00431] ⌈In the Dlt_DetForwardErrorTrace () function a Dlt log-message shall be build and send by calling internally the Dlt_SendLogMessage() function.⌋(SRS_Dlt_00006)

[SWS_Dlt_00376] ⌈The ApplicationID, ContextID and MessageID of the send log message shall have the following values:

ApplicationID = “DET”
ContextID = “STD”
MessageID = 0x00000002

⌋()

[SWS_Dlt_00480] ⌈The log message generated in the function Dlt_DetForwardErrorTrace shall have the following payload entries:

Number	*Type*	*Name*	*Description*
1	uinit16	ModuleId	see Det_ReportError() in [11]
2	uint8	InstanceId	
3	uint8	ApiId	
4	unit8	ErrorID	

Table 7-4 The payload attached the log message generated Dlt_DetForwardErrorTrace (See 7.7.5 Payload, 8.4.2.1 Dlt_SendLogMessage and 8.4.3.1 Dlt_DetForwardErrorTrace)

The meaning of this entries is equivalent to the arguments provided to the Det_ReportError() function specified in Det [11].⌋ ()

7.3.4 Recommendation for generation of Message IDs

The payload of a Non Verbose Message contains the Message ID. The Message ID shall be unique for a ECU. The problem is that Message IDs are provided by a SW-C (the user of Dlt) and at the point in time of the coding of the log and trace message call there is no instance to guarantee the uniqueness of the Message ID.

A possible solution is to map all log messages in a virtual memory segment and then use the memory address as Message ID. Another solution is to have an authoring tool that is responsible for the uniqueness of the Message IDs.
In addition, it could be possible to assign Message ID values in the post build process, so uniqueness for the ECU can be guaranteed
Important is that for every Message ID a description for the associated message is provided.

[SWS_Dlt_00031] ⌈Message IDs used for Dem (0x00000001) and Det (0x00000002) are reserved and not usable for SW-Cs.⌋ ()

7.4 Communication from Dlt with external client

Dlt provides two possible ways to permit an external client the receiving of log and trace message. The communication can be realized by standard diagnostic services, but this is very limited in bandwidth. The alternative is realizing communication over a board specific communication interface, which shall be implemented as a Complex Driver (thus enabling the usage of the standard interfaces like CAN, Flexray or Ethernet).

7.4.1 Communication over standard Dcm channel

One possible communication interface uses standard diagnostic communication over UDS. Dcm [6] provides the access to this service. Dlt shall be aware of using

diagnostic interfaces of Dcm to send log and trace messages (see chapter 7.7) and to send and receive control messages (see chapter 7.7.7.1).
The diagnostic services ReadDataByIdentifier (SID 0x22) and WriteDataByIdentifier (SID 0x2E) shall be used to transport Dlt messages in both direction. (From Dlt to external client and from external client to Dlt). The Dlt messages is completely placed in the data section of these services. Dlt defines its own PDU within this transported data. The DIDs specified by UDS are only used for addressing the Dlt module.

NOTE: Mostly diagnostic service (UDS over the OBD car connector) are very limited in bandwidth, log and trace messages shall be used and transmitted very carefully. The log and trace levels shall be set very limited, to prevent loose of messages.
Dcm provides security mechanisms during production phase which shall be used by Dlt (see chapter 7.5). When an external diagnostic client is connected to the ECU over Dcm, the external client shall set up a “diagnostic session” and hold this active.

The DID (Diagnostic identifier) used within Dlt shall be configured in Dcm. Therefore, the normal configuration parameter of Dcm can be used. Then each ReadDataByIdentifier() and WriteDataByIdentifier () service call is forwarded from Dcm to Dlt. For this purpose, the configuration container “DcmDspDid” of the Dcm module shall be used. There the provided functions of Dlt and the corresponding DID shall be configured.

[SWS_Dlt_00339] ⌈An interface between Dlt and Dcm shall be implemented. This interface shall consist as the following routines described in chapter 8.4.4 provided by Dlt for calling from Dcm:

- Dlt_ReadData
- Dlt_ReadDataLength
- Dlt_WriteData
- Dlt_ConditionCheckRead
- Dlt_ActivateEvent

For sending Dlt control messages from external client to the Dlt modulfe, the external client shall use the WriteDataByIdentifier (SID 0x2E) functionality of UDS.⌋ (SRS_Dlt_00001, SRS_Dlt_00035,)

[SWS_Dlt_00435] ⌈The Dlt_WriteData function provided by Dlt shall be called by Dcm if the WriteDataByIdentifier () service of UDS is requested. The argument “data” contains a complete Dlt control message. This message shall be interpreted by Dlt.⌋ (SRS_Dlt_00035)

7.4.1.1 Using ResponseOnEvent with Dcm

Dlt shall use the ResponseOnEvent (ROE (0x86)) functionality provided by UDS and supported from Dcm for sending log and trace messages.
For this reason the Dcm [6] shall be configured to alow ROE functionality with the ReadDataByIdentifier (SID 0x22) with the Dlt module.
The secence for the ROE is shown in chapter 9.3

[SWS_Dlt_00469] ⌈If an external client enables the ROE for the ReadDataByID (0x22) for the Dlt module, the Dcm module calls the Dlt_ActivateEvent.⌋ ()

[SWS_Dlt_00037] ⌈The Dcm_TriggerOnEvent(eventID) diagnostic service shall be used, so that Dlt can send a message on request, each time there is a new log and trace message in its send buffer.⌋ (SRS_Dlt_00035)

NOTE: Only if the ROE is enabled Dlt is allowed to use the Dcm_TriggerOnEvent() function.

[SWS_Dlt_00340] ⌈Dlt triggers the event by calling the function Dcm_TriggerOnEvent(eventID) of Dcm. The eventID used in this function shall be equal to the eventID provided by the the function call of Dlt_ActivateEvent.⌋ (SRS_Dlt_00035)

[SWS_Dlt_00434] ⌈The Dlt_ReadData function provided by Dlt shall be called by Dcm if the ReadDataByIdentifier() service of UDS is requested. The argument “data” shall contain a complete Dlt message, which Dlt wants to send.⌋ (SRS_Dlt_00035)

[SWS_Dlt_00039] ⌈The messages as described in the Dlt protocol specification (see chapter 7.7) shall be transported over UDS without any change or additional header.

UDS can transport up to 4095 Bytes in one packet. The lower layers of the diagnostic stack are responsible for necessary segmentation and reassembly.⌋ (SRS_Dlt_00002, SRS_Dlt_00035)

7.4.2 Communication over Dlt Communication Module

The alternative communication interface for high bandwidth is the Dlt communication module. Dlt defines an internal interface to a Dlt communication module.

[SWS_Dlt_00040] ⌈The Dlt communication module shall be implemented as a CDD.[2]

Dlt specifies a packet format for transmitting log and trace messages out of a ECU in the chapter protocol specification (see 7.7). This format shall be understood as a high-level protocol. It does not care about the used transport medium and its characteristic. It is up to the system designer to choose or define a proper transport channel.

Possible channels are standard CAN and FlexRay Frames, a Serial Line, a XCP transmission or an IP connection. Dlt specifies the interface to a Dlt communication module, which is responsible for encapsulating the Dlt messages in a communication channel, sending and receiving them to and from a communication interface.

For example, the Dlt Communication Module can add in front of each packet a pattern like "DLT"+0x01 for identifying a send packet on a serial communication line. ⌋ (SRS_Dlt_00001, SRS_Dlt_00034)

In case the "Dlt communication module" shall utilize the AUTOSAR communication stack (e.g. for Can, FlexRay, ...) the CDD module needs to be configurable according to [4] chapter 4.5 CDD module.

[SWS_Dlt_00042] ⌈The Dlt Communication Module is responsible for segmentation and reassembly of the messages. ⌋ (SRS_Dlt_00034)

[SWS_Dlt_00043] ⌈One call of the API of the Dlt communication module for transmitting a log and trace message, encapsulates every time a complete log and trace message.⌋ (SRS_Dlt_00002, SRS_Dlt_00034)

7.5 Security

Dlt is used for testing and diagnostic purposes.

[SWS_Dlt_00044] ⌈During development phase the log and trace communication interfaces may be usable without any security mechanisms.⌋ (SRS_Dlt_00029)

[SWS_Dlt_00465] ⌈To be able to use Dlt also during production phase, security mechanisms shall be implemented. Instead of implementing new security mechanisms, the security mechanisms of Dcm [6] shall be used.
Standard diagnostic channels over Dcm already implements diagnostic sessions and corresponding security levels.
Transmitting log and trace messages shall only be possible during a running diagnostic session.
The required diagnostic security level is configured in Dcm.⌋ (SRS_Dlt_00042)

7.5.1 Securing communication over Dcm

Dcm [6] configures, in which session which diagnostic service can be used. When the diagnostic messages has passed Dcm, the messages are forwarded to Dlt.

[SWS_Dlt_00290] ⌈Log and trace messages and the Dlt control messages shall only be handled during a running diagnostic session except the default session.
As a consequence a diagnostic tester or a diagnostic master must be connected to the ECU, running a non-default session all the time.⌋ (SRS_Dlt_00029)

[SWS_Dlt_00046] ⌈The diagnostic services ResponseOnEvent (0x86), ReadDataByIdentifier (0x22) and WriteDataByIdentifier (0x2E) shall be configured to be enabled in a specific diagnostic session.

A corresponding security level is activated from the diagnostic tester. The security level for call of ReadDataByIdentifier (0x22) and WriteDataByIdentifier (0x2E) for the corresponding DID shall be configured in Dcm.⌋ (SRS_Dlt_00029)

7.5.2 Security for communication over Dlt communication module

The communication over the Dcm diagnostic session has a very good security procedure. Unlike the Dlt communication module sends directly over a given interface and do not care about any security. Therefore, it is very important to **enable** this interface only in a secured connection.

[SWS_Dlt_00048] ⌈The communication over the Dlt communication module shall be disabled per default.⌋ (SRS_Dlt_00029)

[SWS_Dlt_00049] ⌈The enabling of the Dlt communication module shall only be possible by sending a control message in a secured diagnostic session as described in chapter 7.5.1.⌋ ()

[SWS_Dlt_00050] ⌈At development phase the Dlt communication module may be enabled per default.⌋ ()

[SWS_Dlt_00051] ⌈If the communication over the Dlt communication module is enabled there is no restriction to this interface for sending Dlt messages but also for receiving Dlt control messages.⌋ ()

7.6 Runtime management and Implementation

7.6.1 Buffering Messages

[SWS_Dlt_00052] ⌈Dlt shall temporarily store a maximum number of log and trace messages in a local buffer, if no connection to a external client is established.⌋ (SRS_Dlt_00037)

[SWS_Dlt_00341] ⌈This buffer shall store incoming messages from SW-C and BSW modules for transmitting to the Dcm or the Dlt communication module.⌋ (SRS_Dlt_00037)

[SWS_Dlt_00342] ⌈The size of this buffer is configured by the configuration parameter DltMessageBufferSize.⌋ (SRS_Dlt_00037)

[SWS_Dlt_00053] ⌈If the buffer is full the oldest messages in the buffer shall be overwritten to store new incoming messages from SW-Cs or BSW modules.

For this behavior, a ring-buffer is recommended. The oldest messages in this case are lost.⌋ ()

[SWS_Dlt_00297] ⌈If message loose happens the Dlt control message MessageBufferOverflow shall be send. Additionally an internal flag shall be set for remembering this messages loose.⌋ ()

7.6.2 Bandwidth management

[SWS_Dlt_00054] ⌈Dlt shall implement a traffic shaping for its communication interfaces (over Dcm interface [6] and over Dlt communication module).⌋ (SRS_Dlt_00030)

[SWS_Dlt_00055] ⌈The configured parameter DltBandwidthForDiagChannel and DltBandwidthForComModule are to use for limiting the bandwidth of the according channels.⌋ (SRS_Dlt_00030)

[SWS_Dlt_00056] ⌈Traffic shaping shall be implemented as an integral addition of transmitted bits in relation to the passed time. The configuration parameter DltTimePeriodTrafficShaping specifies the time for adding traffic from the past (Time window for integral).⌋ (SRS_Dlt_00030)

NOTE: For traffic shaping a sliding time window should be used. This means that a time window for calculating the transmitted data in the past shall be taken. For example this can be done by adding all transmitted data within the last 10 seconds (time window) to calculate the total transmitted data volume (e.g 10 kbit) within the time window. Then this volume shall be divided through 10 seconds to get the used bandwidth (in this case 1 kbit per second). This time window is sliding, this means that every time the used bandwidth is checked the traffic of the last 10 seconds (the time window) should be analyzed.

[SWS_Dlt_00344] ⌈If the bandwidth with in this window is too high Dlt shall add additional delays before it can send new messages over its interfaces.⌋ (SRS_Dlt_00030)

7.6.3 Interfaces and behavior of Dlt communication module

[SWS_Dlt_00461] ⌈The Dlt communication module shall have the interfaces specified by chapter 8.6.2. ⌋(SRS_Dlt_00001, SRS_Dlt_00034)

[SWS_Dlt_00462] ⌈The Dlt core module shall provide the interfaces specified in chapter 8.4.5⌋(SRS_Dlt_00034)

[SWS_Dlt_00463] ⌈The concert realization of the Dlt communication module is implementation specific, but it shall meet the specified behavior of its interfaces. ⌋(SRS_Dlt_00034)

7.6.4 Administration of pairs of Application ID and Context ID, log levels and trace status

Each SW-C/runnable shall register its used Application IDs and Context IDs.

[SWS_Dlt_00057] ⌈Dlt shall be aware of the registered Application- and Context IDs and store it internally as long as the ECU is running.

At runtime the log levels of different Application IDs and Context IDs can be changed by an external client. Dlt shall manage this runtime configuration.⌋()

7.6.4.1 Port Defined Argument Values and LogTraceSessionControl port interface

For every function call of Dlt_SendLogMessage, Dlt_SendTraceMessage and Dlt_RegisterContext a Port Defined Argument Value is associated with the corresponding Port. This Port Defined Argument Value is called within the Dlt “Session ID”. It defines a connection of a specific log or trace endpoint like a port interface of a runable (service need).

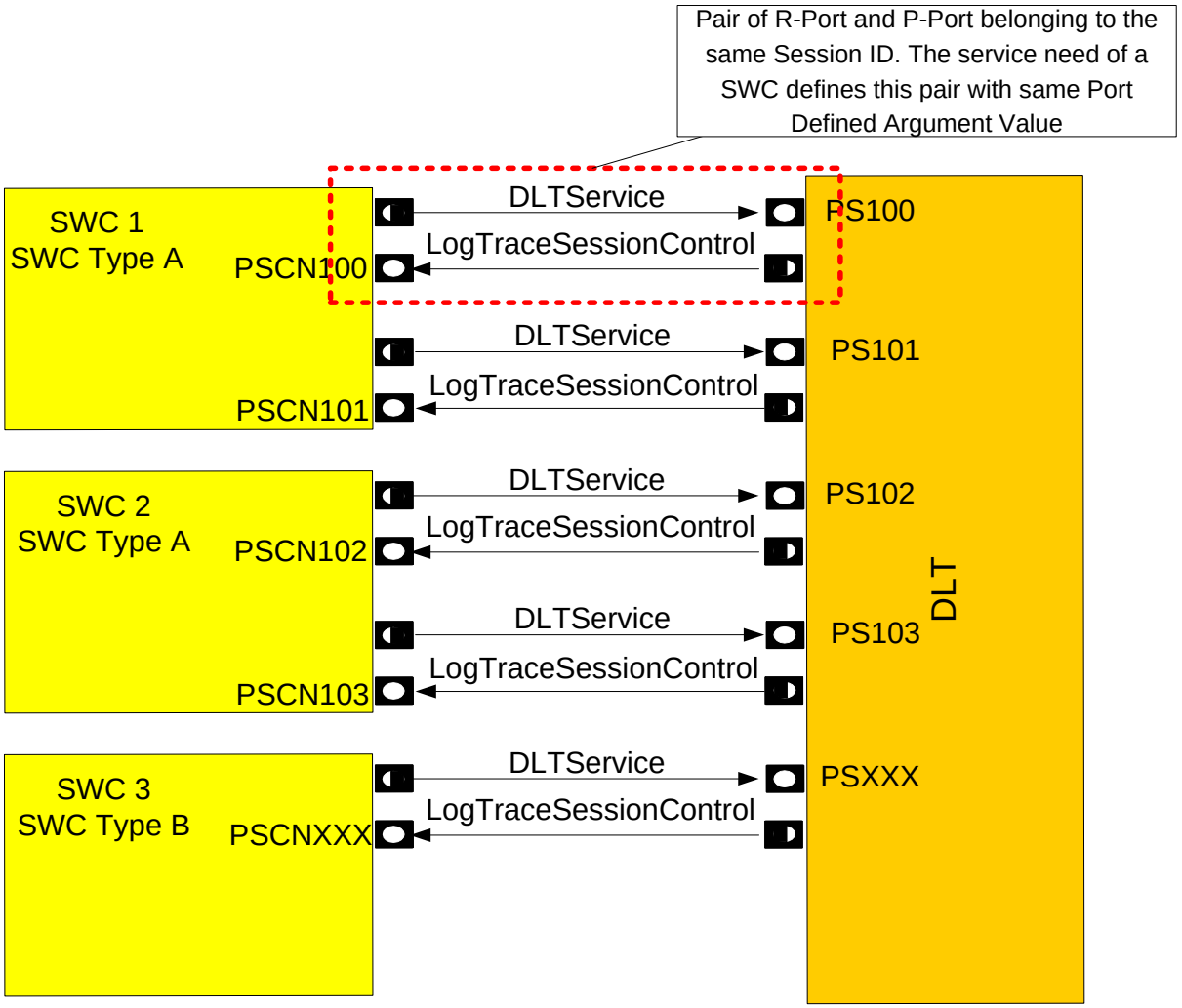


Figure 16 Logical view of R-Port and P-Port connections between SW-Cs and Dlt. SW-C uses the P-Port DLTService of Dlt for sending log or trace messages. Dlt tells the SW-Cs changes about the log level over the P-Port LogTraceSessionControl of the SW-C. DLTService Interface provides a port defined argument value to identify the sender of messages.

The communication between SW-Cs is in some kinds different in modeling and implementing as the communication between SW-Cs and BSW modules. Because of that, when a client server interface between the SW-C and a BSW module is used, some differences are to manage.

If the Dlt provides a port, it is the very same procedure like between SW-Cs. The SW-C calls the provide function from the RTE. The RTE translates the call and forwards it to the P-Port (the Dlt module). Because Dlt provides only one C-function the RTE adds the port defined argument value. Than Dlt can distinguish between the different sources (R-Ports of the SW-Cs).

In the other direction (if Dlt wants to call a P-Port of a SW-C) it is a little bit more complicated. Because of the RTE dose not multiplex the connection (as it did de-multiplex by calling from different SW-Cs) Dlt has to do the multiplexing functionality (see Figure 17). For every P-port of a SW-C, Dlt wants to notify, Dlt hast to call a separate function provided by the RTE.

At the time the Dlt module is to be build or generated it has to know all communication partners. This partners are the SW-Cs which define a R-port and a P-port interface for use with the Dlt. This ports are provided within the SW-C description. In this document the „ServiceNeeds“ holding the information which ports are required from Dlt or provided to Dlt. Also the „ServiceNeeds“ holds the information which pair of DLTService and LogTraceSessionControl ports belonging together. These ports shall have the same port defined argument value which is later on used by Dlt as Session ID.

If the Dlt module is up to generate, all SW-C descriptions of the ECUs SW-C shall be scanned for according ports and collect all „ServiceNeeds“. Than with this information a service component description for the Dlt module shall be generated. This can be done manually or by an automated process. The service component description for the Dlt is an equivalent to the SW-C description and used by the RTE to generate all needed functions.

In the „ServiceNeeds“ of the SW-Cs the pairs of DLTService and LogTraceSessionControl ports are given with their corresponding port defined argument value. This information shall be ported to the Dlt SW-C description. From this description the information about the Session IDs and the corresponding functions in the RTE can be extracted (see Figure 16).

[SWS_Dlt_00058] ⌈The Port defined Argument Values shall be used for identifying a session used on an ECU.⌋ (SRS_Dlt_00038)

[Info]:
The port defined argument value corresponds to the defined Session ID (in this document). The value shall start at 0x1000 (for BSW modules the module ID is taken, starts at 0x0). For connecting a Dlt Service Port to a SW-C the port defined argument value is incremented every time. Therefore, the value is of 0x1000 + n, where n is a continuous number.

[SWS_Dlt_00059] ⌈Every SW-C/runnable which wants to use the Dlt Service shall provide a LogTraceSessionControl client server interface.

This interface is for telling the runnable the new log levels or trace status by Dlt.⌋ (SRS_Dlt_00033, SRS_Dlt_00038)

[SWS_Dlt_00289] ⌈Dlt shall generate for every service need from a SW-C the function calls for all corresponding LogTraceSessionControl interfaces on the RTE (compare Figure 17). ⌋ (SRS_Dlt_00005)

[SWS_Dlt_00060] ⌈Dlt shall handle a table which holds the SessionIDs and the pointers to the interface functions.

SessionID (Port Defined Argument Value)	Pointer to interface function for **LogTraceSessionControl** interface
0x1001	Rte_Call_PSCN001_Dlt_SetLogLevel
	Rte_Call_ PSCN001Dlt_SetTraceStatus
0x1002	Rte_Call_ PSCN002_Dlt_SetLogLevel
	Rte_Call_ PSCN002_Dlt_SetTraceStatus
…	…

Table 7-5 Table which SessionIds and interface functions to hold by Dlt

⌋()

[SWS_Dlt_00345] ⌈The prototypes for the functions to be stored in the table above shall be taken from the Dlt SW-C description.⌋ (SRS_Dlt_00005)

[SWS_Dlt_00426] ⌈The Session IDs and the connection to the corresponding functions shall be taken form the Dlt SW-C description. The Session ID corresponds to the port defined argument value given in the Dlt SW-C description.⌋ (SRS_Dlt_00005)

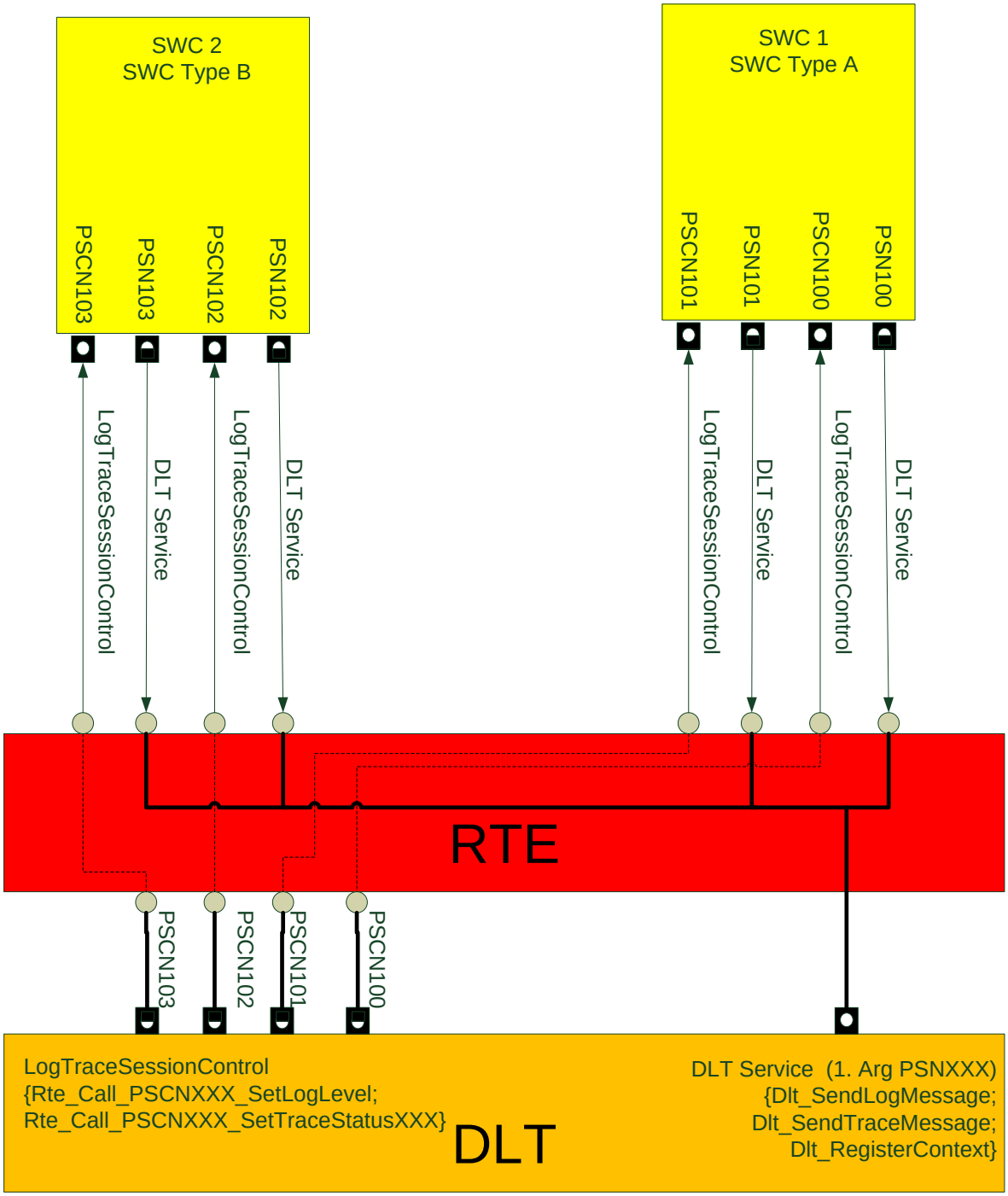


Figure 17 View of a programmer for communication between Dlt and SWS over the RTE. The port interface DLTService is forwarded by the RTE from SW-C to Dlt (by adding a port defined argument value). If Dlt wants to call the LogTraceSessionControl port of a SW-C it shall call a function provided by the RTE (Rte_Call_XXX), then RTE forwards the call to the SW-C P-Port.

7.6.4.2 Configuration and usage of Dlt ServiceNeeds

The “SoftwareComponentTemplate” [8] specifies for the communication between SW-Cs and BSW modules so called “ServiceNeeds”. An instance of the class

“ServiceNeeds” is referenced by the “SwcServiceDependency”. The SoftwareComponentTemplate specifies a meta class called “DltUserNeeds”. For the use of a specific port interface with Dlt by a SW-C for each used SessionID a new instance of “SwcServiceDependency” shall be referenced by the “SwcInternalBehavior” instance of the SW-C. This attached “SwcServiceDependency” shall reference a “ServiceNeeds” class which shall be derived from the class “DltUserNeeds”.

[Requirement for SW-C configuration]
The SW-C description shall be build as follows:
For each group of ports which belong to one SessionID and shall be handled with one PortDefinedArgumentValue by the Dlt service:

- For each used SessionID create one "SwcServiceDependency" as part of the "SwcInternalBehavior"
- Add the "DltUserNeeds" to this "SwcServiceDependency"
- For each included Port add one "RoleBasedPortAssignment" with a reference to the "PortPrototype"
- The role of "RoleBasedPortAssignment" can be left empty
- Create a new “PortAPIOption” with the value of the SessionID as “PortDefinedArgumentValue”
- Attach to "RoleBasedPortAssignment" all “PortPrototype” elements which shall belong to this SessionID

[workflow for Dlt generation tool]
At the generation phase the generation tool of Dlt shall scan the SW-C-description files of the SW-Cs. There it shall perform the following steps to generate the dependencies of SessionID and assigned port interfaces.
Go thru all "SwcInternalBehavior" instances of a SW-C and search the "SwcServiceDependency" classes which contain a "DltUserNeeds"
Go thru all attached "RoleBasedPortAssignments" and create a list of all attached "PortPrototype" elements.
Find for all found PPorts in the "PortPrototype" the "PortDefinedArgumentValue". This can be done by searching the "PortAPIOption" elements which belong to the "InternalBehavior"
Fill the tables described in 7.6.4.3 with the references to the PPorts of the SW-Cs and the belonging SessionIDs
Generate the SW-C-description file for the Dlt module which contain the RPort matching the PPorts found at the SW-Cs.

[SWS_Dlt_00471] ⌈At generation phase the Dlt generation tool shall scan the SW-C description files from the SW-Cs running on the local ECU (see ECU Configuration Specification [4]).⌋ ()

[SWS_Dlt_00472] ⌈With the information from the SW-C description files of the SW-Cs Dlt shall generate its own SW-C-description file for specifying the provided port interfaces.⌋ ()

7.6.4.3 Recommended tables in Dlt to hold information about Application and Context IDs

Figure 18 shows a solution with four tables. This is for a large number of Application and Context IDs. Here a hierarchical search can be done. The size (number of lines) of table (i) corresponds to DltMaxCountAppIds. The number of rows of table (ii) corresponds to DltMaxCountContextIdsPerAppId. The number of lines of table (iii) corresponds to DltMaxCountContextIds. Table (iv) is generated from the „ServiceNeeds“ of all connected SW-Cs/runnables and the corresponding Port Defined Argument Value.

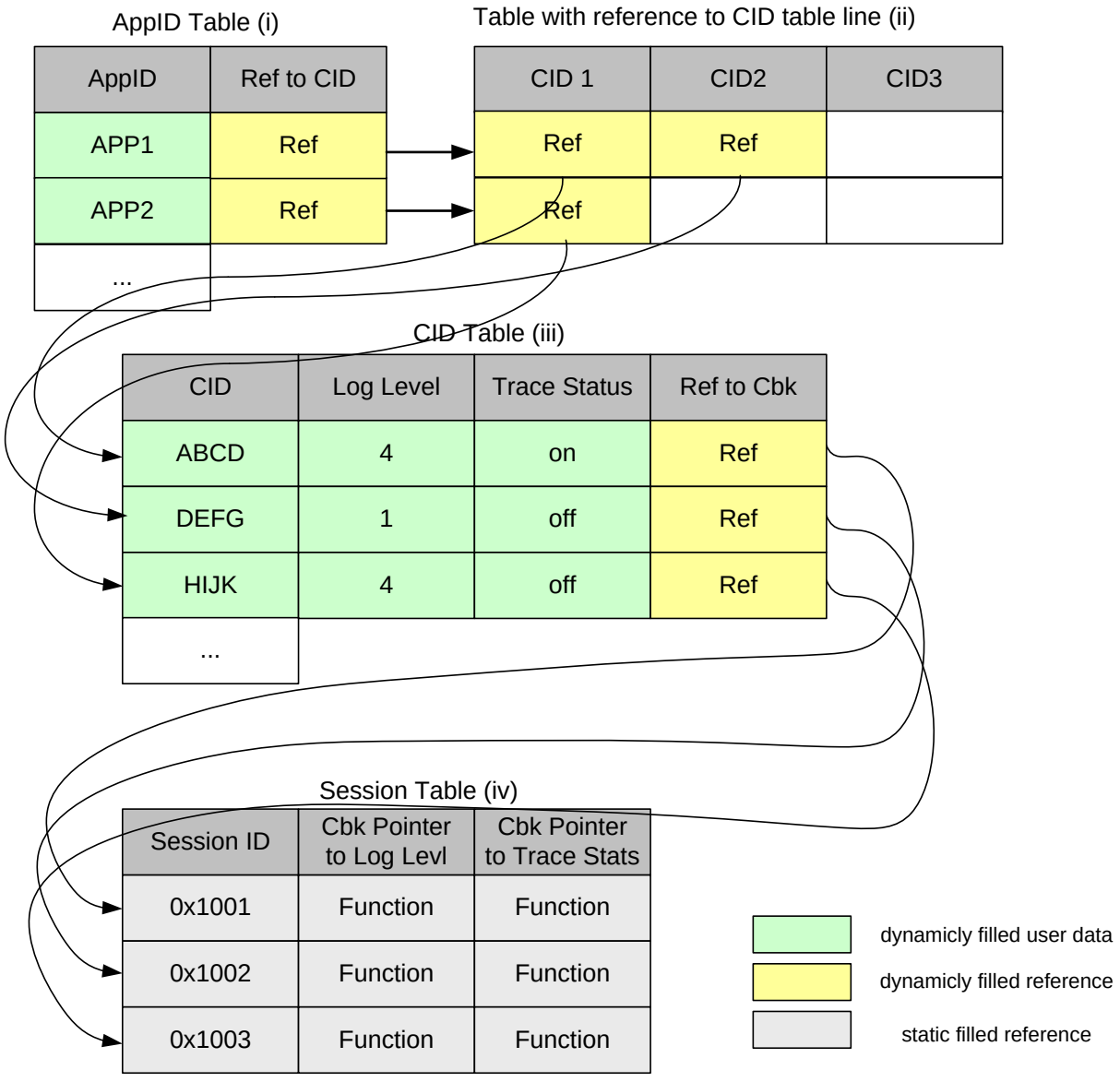


Figure 18 For large numbers of pairs of Application IDs and Context IDs several tables can be used for holding the information. For a faster search table (i) holds the Application IDs and references to an array (ii) which holds references to the Context IDs with additional information. So a two step search can be done. (CID == Context ID; AppID == Application ID)

Figure 19 shows a solution with two tables. This is for a smal number of Application and Context IDs. Here a linear search must be done. The size (number of lines) of table (i) corresponds to DltMaxCountAppIds. The number of rows of table (ii) corresponds to DltMaxCountContextIds. Table (ii) is generated form the „ServiceNeeds“ of all connected SW-Cs/runnables and the corresponding Port Defined Argument Value.

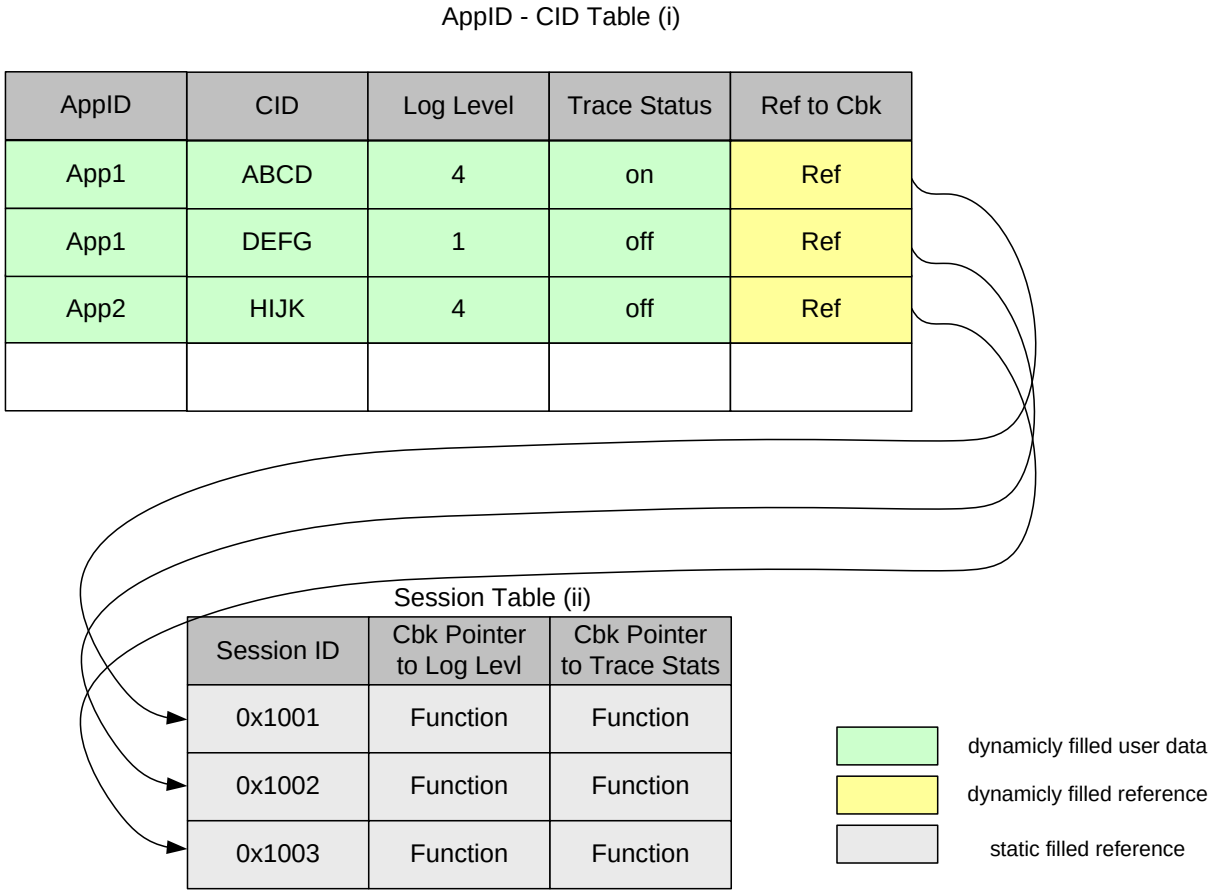


Figure 19 Alternatively a more simple can be used where only one table holds information of Application ID and Context ID. But here a linear search for all pairs is needed. (CID == Context ID; AppID == Application ID)

[SWS_Dlt_00061] ⌈The contents of the described tables (except the Session Table) shall be the contents to store in the NVRAM, if a persistent storage of the configuration of log levels and trace status is requested.

The implementation and representation of the tables can be done in a different way than the described tables here, but the function shall be the same.⌋ ()

[SWS_Dlt_00064] ⌈If DltImplementAppIdContextIdQuery is true additionally to each Application ID and Context ID a description string shall be storable.⌋ (SRS_Dlt_00033)

7.6.5 Message Filtering

Dlt shall check if the log level of the incoming log message is the same or below as the maximum log level stored for this Context ID - Application ID pair (the log level of the incoming message shall be in the pass through range).
If the check is not successful, the messages shall be discarded, otherwise the message shall be transmitted to the external client.

[SWS_Dlt_00065] ⌈If DltImplementFilterMessages is enabled and DltUseFilterMessages is set, Dlt shall filter all incoming log and trace messages.⌋ (SRS_Dlt_00040)

[SWS_Dlt_00066] ⌈If DltImplementFilterMessages is not enabled all functionality for filtering messages in Dlt from SW-Cs and BSW modules shall be left.⌋ ()

[SWS_Dlt_00067] ⌈If no explicit log level or trace status is set the value of DltDefaultMaxLogLevel shall be used instead.⌋ (SRS_Dlt_00040)

[SWS_Dlt_00068] ⌈Also for every Context ID / Application ID pair the status of the trace status shall be stored in Dlt.⌋ (SRS_Dlt_00031, SRS_Dlt_00040)

[SWS_Dlt_00347] ⌈If the trace status for a pair is disabled, the incoming trace messages shall be discarded, even if a connection to an external client is available.⌋ (SRS_Dlt_00040)

7.6.5.1 Administration of log level for filtering

[SWS_Dlt_00069] ⌈If the maximum log level for a Application ID and Context ID is changed by an external client at runtime, Dlt shall store this changes in the corresponding tables. ⌋ (SRS_Dlt_00031)

NOTE: These tables are in RAM, that means that storing in the table is not persistent after next startup. Additionally SW-Cs shall be informed of the log level change (see 7.3.3.1.4).

[SWS_Dlt_00070] ⌈At startup the parts of the table which are changeable at runtime shall be restored from stored data in NVRAM.⌋ ()

7.6.5.2 Administration of trace state for filtering

[SWS_Dlt_00071] ⌈If the trace status for a Application ID and Context ID is changed by an external client at runtime, Dlt shall store this changes in the corresponding tables.⌋ (SRS_Dlt_00031, SRS_Dlt_00040)

NOTE: These tables are in RAM that means that storing in the table is not persistent after next startup. Additionally SW-Cs shall be informed of the trace status change (see 7.3.3.1.4).

[SWS_Dlt_00072] ⌈At startup the parts of the table which are changeable at runtime shall be restored from stored data in NVRAM.⌋ ()

7.6.6 Storing Configuration in NVRAM

[SWS_Dlt_00287] ⌈If the configuration parameter DltImplementNVRamStorage is set, Dlt shall implement the possibility to store some initial values of runtime variables persistent.

The Block ID in the NVRam module [12] shall be DltNvramBlockId.⌋ (SRS_Dlt_00039)

[SWS_Dlt_00073] ⌈If DltImplementNVRamStorage is enabled the log levels and trace status, which are explicitly set for a pair of Application ID and Context ID by an External Client at runtime, shall be storable persistent.⌋ (SRS_Dlt_00039)

[SWS_Dlt_00074] ⌈If DltImplementNVRamStorage is enabled the information about enabling or disabling any interfaces of the Dlt communication module shall be stored persistent.⌋ (SRS_Dlt_00039)

[SWS_Dlt_00076] ⌈If DltImplementNVRamStorage is enabled the bandwidth adjustments shall be storable persistent.⌋ (SRS_Dlt_00039)

[SWS_Dlt_00077] ⌈Dlt shall have a runtime variable for the following configuration parameters:

- DltFilterMessages
- DltDefaultMaxLogLevel
- DltHeaderUseTimestamp
- DltHeaderUseEcuId
- DltHeaderUseExtendedHeader
- DltHeaderUseSessionId
- DltHeaderUseVerboseMode
- DltBandwidthForDiagChannel
- DltBandwidthForComModule
- DltVfbTraceLogLevel
- DltDefaultTraceStatus

to allow a reconfiguration at runtime.⌋ (SRS_Dlt_00031, SRS_Dlt_00039)

NOTE: The runtime variables required by SWS_Dlt_00077 shall be used in the implementation instead of directly accessing the configuration parameters.

[SWS_Dlt_00451] ⌈If DltImplementNVRamStorage is enabled, non-volatile memory blocks (configurable in size by the NVRAM module) shall be used by the Dlt module to achieve permanent storage of variables values required in SWS_Dlt_00077.⌋ ()

[SWS_Dlt_00449] ⌈If DltImplementNVRamStorage is enabled, the Dlt module has to verify the validity of its non volatile blocks.⌋ ()

[SWS_Dlt_00350] ⌈If DltImplementNVRamStorage is enabled the value of the configuration parameter from SWS_Dlt_00077 are to understand as the initial value for the data in the NVRAM.⌋ ()

NOTE: Initial values in this case are the initial values for the persistent stored values for the first startup of the ECU.

[SWS_Dlt_00078] ⌈The storing of information to NVRAM memory RAM blocks shall only be done when the external client requests the storing persistently of this data and if DltImplementNVRamStorage is enabled.⌋ ()

[SWS_Dlt_00452] ⌈If DltImplementNVRamStorage is enabled the Dlt module shall use the API NvM_WriteBlock of the NVRAM module for persistent storing.

If this explicit store request is not done, Dlt restores the untouched NVRAM data at next ECU startup in the Dlt_Init function.⌋ (SRS_Dlt_00039)

[SWS_Dlt_00453] ⌈If DltImplementNVRamStorage is enabled the Dlt module shall use the API NvM_ReadBlock of the NVRAM module for restoring the values from persistent storage for the variables required by SWS_Dlt_00077.⌋ (SRS_Dlt_00039)

[SWS_Dlt_00491] ⌈The restoring of the parameters mentioned in **[SWS_Dlt_00453]** shall be done in the Dlt_Init() function.⌋ ()

[SWS_Dlt_00450] ⌈After the API Dlt_Init the Dlt shall be fully operational.⌋ ()

[SWS_Dlt_00288] ⌈If DltImplementNVRamStorage is not set, persistent storage shall not be used.

Runtime variables shall be used to allow reconfiguration at runtime. The different is that this configuration is not persistent stored into NVRAM and the defaults are restored at ECU startup.

If requested, a factory default of log level and trace status of all Context IDs and Application IDs shall be set.⌋ (SRS_Dlt_00039)

[SWS_Dlt_00348] ⌈Reset to factory default shall be done by deleting the individual settings for Application IDs and Context IDs and setting the maximum default log level to DltFactoryDefaultMaxLogLevel. Also the reset to factory default shall set the initial values for the variables required in SWS_Dlt_00077 to the values of the corresponding configuration parameters.⌋ ()

7.6.7 Processing of control messages

Dlt uses control messages for reconfiguration at runtime (see chapter 7.7.6.1). Control messages are mostly send by an external client and interpreted by Dlt. Afterwards Dlt sends control messages as answer back to the external client.

[SWS_Dlt_00079] ⌈Dlt shall process control messages. It shall receive these messages, process it (by doing an accurate action) and response to the request.

The response is also a normal Dlt message, which is to place in the send-buffer.⌋ (SRS_Dlt_00031)

[SWS_Dlt_00351] ⌈The size of the generated control messages to send shall not exceed DltMaxMessageLength.⌋ ()

7.6.8 Message Handling

If Dlt receives a message from SW-Cs or BSW modules by a call to Dlt_SendLogMessage or Dlt_SendTraceMessage the following procedure shall be performed.

[SWS_Dlt_00080] ⌈Dlt shall copy the provided payload to the send buffer of Dlt. In most cases the payload provided by a SW-C or BSW module is attached to a message without modification. (exception see 7.6.8.3)⌋ ()

[SWS_Dlt_00298] ⌈Dlt shall add the message header (see 7.7) for transmitting the message over the network. The content of the header depends on the provided information by the call of Dlt_SendLogMessage and Dlt_SendTraceMessage and on some configuration parameters.⌋ ()

[SWS_Dlt_00081] ⌈If the message length exceeds DltMaxMessageLength the message shall be discarded and the call of Dlt_SendLogMessage and Dlt_SendTraceMessage shall return with DLT_E_MSG_TOO_LARGE.⌋ ()

7.6.8.1 Filling the Header

As described in 7.7 Dlt uses a protocol for transmitting messages. If a log or trace message is received by Dlt some entries of this protocol shall be filled.

[SWS_Dlt_00082] ⌈Table 7-6 shows the connection between the configuration parameters and equivalent bit entries in the field header type (HTYP), which Dlt shall implement.

Protocol parameter	Configuration parameter
Use Extended Header (UEH)	DltHeaderUseExtendedHeader
MSB First (MSBF)	DltHeaderPayloadEndianes
With ECU ID (WEID)	DltHeaderUseEcuId
With Session ID (WSID)	DltHeaderUseSessionID
With Timestamp (WTMS)	DltHeaderUseTimestamp
Version Number (VERS)	Version number of Dlt protocol used by this Dlt implementation
ECUID (ECU)	DltEcuId

Table 7-6 Header Type (HTYP) bit entries in dependency on configuration parameters.

⌋()

[SWS_Dlt_00083] ⌈The related entries in the header TMSP and ECU shall be done in dependency to the bits set in the HTYP.
The timestamp shall be generated at the moment the Dlt_SendXXXMessage was called. It shall be the local time from the ECU (uptime). One hardware free running timer (HWFRT) of the AUTOSAR GPT module can be used to get a timestamp.⌋()

[SWS_Dlt_00084] ⌈The field Message Counter (MCNT) shall be incremented for every message which is put to the send buffer (see 7.6.1).⌋(SRS_Dlt_00018)

[SWS_Dlt_00085] ⌈The field Length (LEN) shall contain the overall length of the send message in byte.
The field Session ID (SEID) shall be filled with the Session ID (Port Defined Argument Value) provided by the call off Dlt_SendLogMessage and Dlt_SendTraceMessage.⌋()

7.6.8.2 Filling the extended Header

[SWS_Dlt_00086] ⌈The extended Header shall only be attached when the UEH flag is set. The information for the extended header shall be taken from the log_info/trace_info parameter from the function Dlt_SendLogMessage/Dlt_SendTraceMessage.⌋(SRS_Dlt_00019)

[SWS_Dlt_00087] ⌈The functionality for filling the extended header can be left if DltImplementExtendedHeader is not set.⌋ ()

[SWS_Dlt_00088] ⌈

Protocol parameter	Description of source
Verbose (VERB)	log_info.options.verbose_mode
Message Type (MSTP)	DLT_TYPE_LOG if call to Dlt_SendLogMessage DLT_TYPE_APP_TRACE if call to Dlt_SendTraceMessage DLT_TYPE_NW_Trace if call from RTE trace
Number of arguments (NOAR)	log_info.options.arg_count
Application ID (APID)	log_info.app_id
Context ID (CTID)	log_info.context_id

Table 7-7 Fields of the extended Header and how to fill them

⌋ (SRS_Dlt_00019)

7.6.8.3 Switch between Verbose and Non Verbose Mode

[SWS_Dlt_00089] ⌈Normally the payload of a Dlt message is passed without modification. It is up to the SW-C to manage the payload in Verbose or Non Verbose Mode.⌋ ()

[SWS_Dlt_00090] ⌈If DltImplementVerboseMode is not true, a call to Dlt_SendLogMessage or Dlt_SendTraceMessage shall return DLT_E_NOT_IN_VERBOSE_MODE to the caller of the function and reject the message if it sends a message in Verbose Mode (verbose mode flag set).⌋ ()

7.7 Protocol Specification (for transmitting to a external client and saving on the client)

Dlt translates and serializes the messages transferred from the user API into a byte stream.This protocol specification describes the format of this byte stream. The stream data can also be saved in a file in the external client.

The protocol supports a verbose and a non-verbose mode. In the verbose mode, the complete description of the transferred data is provided within the protocol. The result is a self-describing data format. In the non-verbose mode, only data of non-static information is transmitted (see 7.7.5.1) and the description is provided externally.

[SWS_Dlt_00300] ⌈Depending on the configuration parameter DltUseVerboseMode, the protocol shall support verbose or non-verbose mode.⌋ ()

7.7.1 Dlt Message Format in General

The byte stream consists of one or more Dlt messages that are ordered back-to-back, without any separation marks. One Dlt Message consists of a mandatory Standard Header, which contains essential information for processing the message, of an optional Extended Header, which provides detailed information about the message and of optional payload.
Reducing the size of the Standard Header by skipping optional fields have advantages if bandwidth is very limited.

[SWS_Dlt_00301] ⌈The Dlt message shall consist at least of a Standard Header.⌋ (SRS_Dlt_00002, SRS_Dlt_00013)

[SWS_Dlt_00467] ⌈Following table (Table 7-8) shows a general assembly of one Dlt message. Every Dlt message shall consist of the shown entries.

Length (bytes)	Name	Description
4,8,12 or16	**Standard Header (Mandatory)**	Contains essential information for interpreting the Dlt message
10	**Extended Header (Optional)**	Can be added optionally for providing more information or for use in control messages
x	**Payload (Optional)**	Contains information about a specific log and trace message

Table 7-8 General Dlt message format

⌋ (SRS_Dlt_00002)

7.7.2 Header Definition of the Dlt Protocol

[SWS_Dlt_00091] ⌈The Standard Header and the Extended Header shall be in big endian format (MSB first).⌋ ()

7.7.3 Standard Header

[SWS_Dlt_00302] ⌈The Standard Header shall be at the beginning of a Dlt Message.

The following table gives an overview of the composition of the Standard Header. Detailed description of the entries follows.⌋ (SRS_Dlt_00002, SRS_Dlt_00013)

[SWS_Dlt_00458] ⌈The Standard Header shall consist of the following entries:

Position in bytes	Number of bits	Name	Short Description
0	8	Header Type (HTYP)	
	bit 0	Use Extended Header (UEH)	If set, the Extended Header is transmitted. If not set, the Extended Header is not transmitted and the message is in non-verbose mode.
	bit 1	MSB First (MSBF)	If set, the payload data is in big endian format, else in little endian format.
	bit 2	With ECU ID (WEID)	If set, the ECU ID (ECU) is attached in the Standard Header.
	bit 3	With Session ID (WSID)	If set the Session ID (SEID) is attached in the Standard Header.
	bit 4	With Timestamp (WTMS)	If set, the timestamp (TMSP) is attached in the Standard Header.
	bit 5-7	Version Number (VERS)	Version number of Dlt Data protocol
1	8	Message Counter (MCNT)	Continuous number of message, for detection of lost messages. Counter is increased for every received message by the Dlt API message in local buffer.
2-3	16	Length (LEN)	Length of the complete message in bytes
Optional if WEID is set			
4-7	32	ECU ID (ECU)	Unique address of sender (Diag-Addr / ECU Name / ...), interpreted as 4 ASCII characters
Optional if WSID is set			
8-11	32	Session ID (SEID)	Session number
Optional if WTMS is set			
12-15	32	Timestamp (TMSP)	Continuous time / ticks from the ECU at the moment the message is sent to Dlt.

Table 7-9 Overview of the Standard Header

⌋(SRS_Dlt_00016, SRS_Dlt_00017, SRS_Dlt_00018, SRS_Dlt_00022)

7.7.3.1 Header Type (HTYP)

[SWS_Dlt_00094] ⌈The Header Type shall be interpreted as an 8-bit field. The included bits are UEH, MSBF, WEID, WSID, WTMS, VERS (3 bit), as shown in following table:

	Bit Position							
Offset	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	UEH	MSBF	WEID	WSID	WTMS	VERS	VERS	VERS

Table 7-10 Assembly of the Header Type in Standard Header

⌋()

7.7.3.1.1 Use Extended Header (UEH)

[SWS_Dlt_00406] ⌈The Use Extended Header (UEH) bit is set depending on the configuration parameter DltHeaderUseExtendedHeader.
If it's set, Extended Header (see 7.7.4) adjoins the Standard Header else the Extended Header is skipped.⌋()

[SWS_Dlt_00095] ⌈If the UEH bit is set, the Extended Header shall be transmitted after the Standard Header.⌋()

[SWS_Dlt_00303] ⌈If the UEH bit is not set, the Extended Header shall not be transmitted after the Standard Header.⌋(SRS_Dlt_00044)

[SWS_Dlt_00096] ⌈If the UEH bit is not set, the payload shall be interpreted as in non-verbose mode.⌋(SRS_Dlt_00024)

7.7.3.1.2 Most Significant Byte First (MSBF)

The MSBF bit specifies the byte order of the payload. It depends on the configuration parameter DltHeaderPayloadEndianes.

[SWS_Dlt_00097] ⌈If the MSBF bit is set, the most significant byte shall be first in payload (big endian format).⌋(SRS_Dlt_00014, SRS_Dlt_00016)

[SWS_Dlt_00304] ⌈If the MSBF bit is not set, least significant byte shall be first in payload (little endian format).⌋(SRS_Dlt_00014, SRS_Dlt_00016)

7.7.3.1.3 With ECU ID (WEID)

With this parameter the sender of a message can be identified unique. The WEID bit is set depending on the configuration parameter DltHeaderUseEcuId.

[SWS_Dlt_00098] ⌈If the WEID bit is set, the ECU ID (ECU) shall be transmitted.⌋ (SRS_Dlt_00022)

[SWS_Dlt_00305] ⌈If the WEID bit is not set, ECU ID (ECU) field shall be skipped and is not located in the Standard Header.⌋ ()

7.7.3.1.4 With Session ID (WSID)

[SWS_Dlt_00407] ⌈The WSID bit is set depending on the configuration parameter DltHeaderUseSessionID.⌋ ()

[SWS_Dlt_00101] ⌈If the WSID bit is set, the Session ID shall be transmitted.⌋ (SRS_Dlt_00020)

[SWS_Dlt_00306] ⌈If the WSID bit is not set, the Session ID field shall be skipped and is not located in the Standard Header.⌋ ()

7.7.3.1.5 With Timestamp (WTMS)

[SWS_Dlt_00408] ⌈The WTMS bit is set depending on the configuration parameter DltHeaderUseTimestamp.⌋ ()

[SWS_Dlt_00102] ⌈If the WTMS bit is set, the timestamp shall be transmitted.⌋ (SRS_Dlt_00017)

[SWS_Dlt_00307] ⌈If the WTMS bit is not set, the timestamp (TMSP) field shall be skipped and is not located in the Standard Header.⌋ ()

7.7.3.1.6 Version Number (VERS)

The sender sets the Version Number of the Dlt Data Protocol in the Standard Header according to the used version. The receiver checks the Version Number and interprets the Dlt message according to the Version Number. Future versions of Dlt Data Protocol may exist.

[SWS_Dlt_00318] ⌈The Version Number of the Dlt Data Protocol consists of 3 bit.⌋ ()

[SWS_Dlt_00103] ⌈The Version Number shall always be set.⌋ ()

[SWS_Dlt_00104] ⌈The receiver of a Dlt message shall check the Version Number and shall only interpret the Dlt message if the Version Number is supported.⌋ ()

[SWS_Dlt_00299] ⌈The Version Number for this Dlt Data Protocol is 0x1.⌋ ()

7.7.3.2 Message Counter (MCNT)

The Message Counter counts Dlt messages received by the Dlt module. With the Message Counter, lost messages can be recognized to a certain level.

[SWS_Dlt_00319] ⌈The Message Counter is an unsigned 8-bit (0-255) integer.⌋ (SRS_Dlt_00018)

[SWS_Dlt_00105] ⌈The Dlt module shall increment the Message Counter by one at every message received via the Dlt API.⌋ (SRS_Dlt_00018)

[SWS_Dlt_00106] ⌈If Message Counter reaches 255, it starts by 0 at the next message.⌋ (SRS_Dlt_00018)

7.7.3.3 Length (LEN)

Length (LEN) includes the Standard Header, the optional Extended Header and the optional payload.

[SWS_Dlt_00320] ⌈Length is a 16-bit unsigned integer.⌋ ()

[SWS_Dlt_00107] ⌈Length (LEN) shall hold the total length of the Dlt message in byte.⌋ ()

7.7.3.4 ECU ID (ECU)

The ECU ID identifies the ECU (see 7.1.5).

[SWS_Dlt_00321] ⌈ECU ID is a 32-bit field interpreted as four 8-bit ASCII characters.⌋ ()

[SWS_Dlt_00108] ⌈ECU ID shall be unique within the used domain.⌋ ()

[SWS_Dlt_00308] ⌈If the ECU ID is shorter than four 8-bit ASCII characters, the remaining characters shall be filled by 0x00.

For instance, it can be a diagnostic address or a name consisting of 4 ACSII characters like “ABS1”.⌋ ()

7.7.3.5 Session ID (SEID)

Session ID is the identification number of a log or trace session (see 7.1.6).

[SWS_Dlt_00322] ⌈Session ID is a 32-bit unsigned integer.⌋ (SRS_Dlt_00020)

[SWS_Dlt_00110] ⌈Session ID shall be the Session ID of the port interface (see 7.3.3.1.7), which sends the message.⌋ (SRS_Dlt_00020)

7.7.3.6 Timestamp (TMSP)

[SWS_Dlt_00323] ⌈Timestamp is a 32-bit unsigned integer.⌋ (SRS_Dlt_00017)

[SWS_Dlt_00112] ⌈The TMSP bit field shall hold the timestamp from the moment SW-C sends the message to Dlt.⌋ (SRS_Dlt_00017)

[SWS_Dlt_00309] ⌈The time resolution is in 0.1 milliseconds.⌋ ()

[SWS_Dlt_00113] ⌈Timestamp shall be the uptime of the ECU.⌋ ()

[SWS_Dlt_00481] ⌈One hardware free running timer (HWFRT) of the AUTOSAR GPT module shall be used to get a timestamp. The configuration parameter DltGptChannel devotes the channel to use within the GPT module.⌋ ()

7.7.3.7 Assembly of Standard Header

		Bit position							
Offset to Standard Header start pos in byte	**Field Name**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	**HTYP**	UEH	MSBF	WEID	WSID	WTMS	VERS	VERS	VERS
1	**MCNT**	MCNT	MCNT	MCNT	MCNT	MCNT	MCNT	MCNT	MCNT
2	**LEN**	LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN
3		LEN	LEN	LEN	LEN	LEN	LEN	LEN	LEN
4	**ECU (optio-nal)**	ECU	ECU	ECU	ECU	ECU	ECU	ECU	ECU
5		ECU	ECU	ECU	ECU	ECU	ECU	ECU	ECU
6		ECU	ECU	ECU	ECU	ECU	ECU	ECU	ECU
7		ECU	ECU	ECU	ECU	ECU	ECU	ECU	ECU
8	**SEID**	SEID	SEID	SEID	SEID	SEID	SEID	SEID	SEID

		Bit position							
Offset to Standard Header start pos in byte	**Field Name**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
9	**(optio-nal)**	SEID	SEID	SEID	SEID	SEID	SEID	SEID	SEID
10		SEID	SEID	SEID	SEID	SEID	SEID	SEID	SEID
11		SEID	SEID	SEID	SEID	SEID	SEID	SEID	SEID
12	**TMSP (optio-nal)**	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP
13		TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP
14		TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP
15		TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP	TMSP

Table 7-11 Assembly of the Standard Header

7.7.4 Dlt Extended Header

The Extended Header will be transmitted if the UEH bit (Use Extended Header) is set in the Standard Header.

[SWS_Dlt_00116] ⌈The Extended Header shall be transmitted in the case of a verbose mode of the payload because it holds information about the Dlt log or trace message like Context ID and Application ID.
In the case of a non-verbose mode this information can be stored in an external file and reassigned (by an external client) with the Message ID which is provided in the payload. ⌋(SRS_Dlt_00013)

[SWS_Dlt_00117] ⌈The Extended Header shall be transmitted in case of transmitting control messages (see chapter 7.7.7.1).

The following table gives an overview of the composition of the Extended Header. Detailed description of the entries follows. ⌋(SRS_Dlt_00002)

[SWS_Dlt_00457] ⌈The Extended Header shall contain these entries

Position in bytes	**Number of bits**	**Name**	**Short Description**
0	8	Message Info (MSIN)	

Position in bytes	Number of bits	Name	Short Description
	bit 0	Verbose (VERB)	If set, a description of the transmitted data is provided within the payload. If not set, this information will be given within a file.
	bit 1-3	Message Type (MSTP)	Enum of following types: DLT_TYPE_LOG 0x0 DLT_TYPE_APP_TRACE 0x1 DLT_TYPE_NW_TRACE 0x2 DLT_TYPE_CONTROL 0x3
		Message Type Info (MTIN) depends on **Message Type (MSTP)**	
	bit 4-7	**Message Log Info (MSLI)** *If* MSTP == DLT_TYPE_LOG	DLT_LOG_FATAL 0x1 DLT_LOG_ERROR 0x2 DLT_LOG_WARN 0x3 DLT_LOG_INFO 0x4 DLT_LOG_DEBUG 0x5 DLT_LOG_VERBOSE 0x6
		Message Trace Info (MSTI) *If* MSTP == DLT_TYPE_APP_TRACE	DLT_TRACE_VARIABLE 0x1 DLT_TRACE_FUNCTION_IN 0x2 DLT_TRACE_FUNCTION_OUT 0x3 DLT_TRACE_STATE 0x4 DLT_TRACE_VFB 0x5
		Message Bus Info (MSBI) *If* MSTP == DLT_TYPE_NW_TRACE	DLT_NW_TRACE_IPC 0x1 DLT_ NW_TRACE _CAN 0x2 DLT_ NW_TRACE _FLEXRAY 0x3 DLT_ NW_TRACE _MOST 0x4 Reserved 0x5-0x7 UserDefined 0x8-0x15
		Message Control Info (MSCI) *If* MSTP == DLT_TYPE_CONTROL	DLT_CONTROL_REQUEST 0x1 DLT_CONTROL_RESPONSE 0x2 DLT_CONTROL_TIME 0x3
1	8	Number of arguments (NOAR)	Number of arguments in the message payload
2-5	32	Application ID (APID)	Number / ID of application – interpreted as 4 ASCII characters

Position in bytes	Number of bits	Name	Short Description
6-9	32	Context ID (CTID)	Unique ID of logging / tracing context – interpreted as 4 ASCII characters

Table 7-12 Overview of the Extended Header

⌋(SRS_Dlt_00019, SRS_Dlt_00020, SRS_Dlt_00021)

7.7.4.1 Message Info (MSIN)

[SWS_Dlt_00118] ⌈Message Info shall be interpreted as an 8-bit field. The included bits are VERB, MSTP (3 bit) and MTIN (4 bit).

	Bit Position							
Offset	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	VERB	MSTP	MSTP	MSTP	MTIN	MTIN	MTIN	MTIN

Table 7-13 Assembly of the Message Info in Extended Header

⌋()

7.7.4.1.1 Verbose (VERB)

The VERB bit indicates, if the payload is transmitted in verbose or in non-verbose mode. If the VERB bit is not set, a description of the transmitted data is provided externally, e.g. in an external file.

[SWS_Dlt_00119] ⌈If the VERB bit is set, the payload shall be transmitted in verbose mode.⌋(SRS_Dlt_00044)

[SWS_Dlt_00310] ⌈If the VERB bit is not set, the payload shall be transmitted in non-verbose mode.⌋(SRS_Dlt_00024)

7.7.4.1.2 Message Type (MSTP)

The value of this field describes the transmitted Dlt message

[SWS_Dlt_00324] ⌈Message Type is a 3-bit unsigned integer. ⌋()

[SWS_Dlt_00120**]** ⌈Message Type shall have one of the following values:

Value	Name	Description
0x0	DLT_TYPE_LOG	The transmitted message is a log message
0x1	DLT_TYPE_APP_TRACE	The transmitted message is a trace of a SW-C or VFB [1].
0x2	DLT_TYPE_NW_TRACE	The transmitted message contains a trace of received or sent network messages.
0x3	DLT_TYPE_CONTROL	The transmitted message is a control message. This message can be sent from and to an ECU. It contains control information for connection management, timing issues and for configuration of the Dlt BSW module.

Table 7-14 Message Types of Dlt messages (MSTP)

⌋()

7.7.4.1.3 Message Type Info (MTIN)

[SWS_Dlt_00325] ⌈Message Type Info is a 4-bit unsigned integer. ⌋()

[SWS_Dlt_00121] ⌈The content of the MTIN field depends on the MSTP field according to the following table:

Message Type (MSTP)	Corresponding Message Type Info (MTIN)
DLT_TYPE_LOG	Message Log Info (MSLI)
DLT_TYPE_APP_TRACE	Message Trace Info (MSTI)
DLT_TYPE_NW_TRACE	Message Bus Info (MSBI)
DLT_TYPE_CONTROL	Message Control Info (MSCI)

Table 7-15 Relation between Message Type Info (MTIN) and Message Type (MSTP)

⌋()

7.7.4.1.4 Message Log Info (MSLI)

[SWS_Dlt_00122] ⌈If MSTP equals DLT_TYPE_LOG, MTIN shall have one of following values:

Value	Name	Description
0x1	DLT_LOG_FATAL	Fatal system errors, should be very rare
0x2	DLT_LOG_ERROR	Errors occurring in a SW-C with impact to correct functionality

[1] Unlike in a log message a trace can only be turned on or off. Trace messages can have additional attributes like function-in/out. Trace may not be enabled in production phase.

Value	Name	Description
0x3	DLT_LOG_WARN	Log messages where a incorrect behavior can not be ensured
0x4	DLT_LOG_INFO	Log messages providing information for better understanding of the internal behavior of a software
0x5	DLT_LOG_DEBUG	Log messages, which are usable only for debugging of a software
0x6	DLT_LOG_VERBOSE	Log messages with the highest communicative level, here all possible states, information and everything else can be logged

Table 7-16 Possible MSLI values

⌋(SRS_Dlt_00019)

7.7.4.1.5 Message Trace Info (MSTI)

[SWS_Dlt_00123] ⌈If MSTP equals DLT_TYPE_APP_TRACE, MTIN shall have one of following values:

Value	Name	Description
0x1	DLT_TRACE_VARIABLE	For tracing the value of a variable
0x2	DLT_TRACE_FUNCTION_IN	For tracing the calling of a function
0x3	DLT_TRACE_FUNCTION_OUT	For tracing the returning of a function
0x4	DLT_TRACE_STATE	For tracing a state of a state machine
0x5	DLT_TRACE_VFB	For tracing RTE Events

Table 7-17 Possible MSTI values

⌋()

7.7.4.1.6 Message Control Info (MSCI)

[SWS_Dlt_00124] ⌈If MSTP equals DLT_TYPE_CONTROL, MTIN shall have one of following values:

Value	Name	Description
0x1	DLT_CONTROL_REQUEST	For sending a request message from an external client to the Dlt module
0x2	DLT_CONTROL_RESPONSE	For answering the request message by the Dlt module with a response message
0x3	DLT_CONTROL_TIME	For keep-alive messages

Table 7-18 Possible MSCI values

⌋()

7.7.4.1.7 Message bus Info (MSBI)

[SWS_Dlt_00125] ⌈If MSTP equals DLT_TYPE_NW_TRACE, MTIN shall have one of following values:

Value	Name	Description
0x1	DLT_NW_TRACE_IPC	Inter-Process-Communication
0x2	DLT_NW_TRACE_CAN	Controller Area Network Bus
0x3	DLT_NW_TRACE_FLEXRAY	FlexRay Bus
0x4	DLT_NW_TRACE_MOST	Media Oriented Systems Transport Bus
0x5 – 0x7	Reserved	Reserved for future use
0x8 – 0x15	User Defined	User Defined settings

Table 7-19 Possible MSBI values

⌋()

7.7.4.2 Number of Arguments (NOAR)

Number of Arguments is number of consecutive parameters in the payload of one Dlt message.

[SWS_Dlt_00326] ⌈Number of Arguments is an 8-bit unsigned integer. ⌋()

[SWS_Dlt_00126**]** ⌈Number of Arguments shall contain in Verbose Mode the number of provided arguments within the payload. In Non Verbose Mode it shall contain 0x0. ⌋()

7.7.4.3 Application ID (APID)

Application ID is an abbreviation of the SW-C/BSW module (see 7.1.7).

[SWS_Dlt_00127] ⌈Application ID is a 32-bit field interpreted as four 8-bit ASCII characters.⌋(SRS_Dlt_00021)

[SWS_Dlt_00312] ⌈If the Application ID is shorter than four 8-bit ASCII characters, the remaining characters shall be filled by 0x00. ⌋()

7.7.4.4 Context ID (CTID)

Context ID is a user defined ID to group log and trace messages (see 7.1.8).

[SWS_Dlt_00128] [Context ID is a 32-bit field interpreted as four 8-bit ASCII characters.] (SRS_Dlt_00021)

[SWS_Dlt_00313] ⌈If the Context ID is shorter than four 8-bit ASCII characters, the remaining characters shall be filled by 0x00.

It represents the context, which the message belongs to. ⌋()

7.7.4.5 Assembly of Extended Header

		Bit position							
Offset to Extended Header start pos in byte	**Field Name**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	**MSIN**	VERB	MSTP	MSTP	MSTP	MTIN	MTIN	MTIN	MTIN
1	**NOAR**	NOAR	NOAR	NOAR	NOAR	NOAR	NOAR	NOAR	NOAR
2	**APID**	APID	APID	APID	APID	APID	APID	APID	APID
3		APID	APID	APID	APID	APID	APID	APID	APID
4		APID	APID	APID	APID	APID	APID	APID	APID
5		APID	APID	APID	APID	APID	APID	APID	APID
6	**CTID**	CTID	CTID	CTID	CTID	CTID	CTID	CTID	CTID
7		CTID	CTID	CTID	CTID	CTID	CTID	CTID	CTID
8		CTID	CTID	CTID	CTID	CTID	CTID	CTID	CTID
9		CTID	CTID	CTID	CTID	CTID	CTID	CTID	CTID

Table 7-20 Assembly of Extended Header

7.7.5 Payload

The payload holds the parameters that will be logged or traced. More precisely the payload consists of the buffer that will be passed in the API containing the parameters to be logged or traced. For detailed information, see 8.4.2 and 8.4.2.4.

The payload adjoins next to the Standard Header or the Extended Header, depending on the UEH bit.

[SWS_Dlt_00314] ⌈If the UEH bit is set, the payload shall adjoin the Extended Header. ⌋(SRS_Dlt_00013, SRS_Dlt_00023)

[SWS_Dlt_00315] ⌈If the UEH bit is not set, the payload shall adjoin the Standard Header.

There are two modes for transmitting the payload – Verbose Mode and Non-Verbose Mode. The bit Verbose (VERB) in the Extended Header specifies which mode is used. ⌋(SRS_Dlt_00013, SRS_Dlt_00023)

7.7.5.1 Non-Verbose Mode

There are two types of data relevant in the Non-Verbose Mode – static data and non-static data, detailed description of the data types follows. Non-static data with its unique identifier ID is transmitted in the payload in Non-Verbose Mode. This unique Message ID (see 7.7.5.1.1) is assigned to the non-static data in order that the receiver can reassign the data. Only the Message ID with the non-static data are transmitted within the Dlt message. Static data is not transmitted in the payload in Non-Verbose Mode.

The assembly of a Dlt message in Non-Verbose Mode is shown in the following table.

[SWS_Dlt_00460] ⌈If non-verbose mode is used the packet format in Table 7-21 shall be used.

Length in Byte	Name	Description
4,8,12 or 16	**Standard Header**	Essential information for interpreting the Dlt message
	Payload	
4	Message ID	Message ID is unique for a specific Dlt message. All static information like parameter name and description and static text are associated to this ID. This information is provided by an external file.
x	Non-Static Data	All non-static information of a Dlt message is transmitted here. Static information is associated with the Message ID and is not transmitted.

Table 7-21 Assembly of a Dlt Message in Non-Verbose Mode

Static data are all data that are not modifiable at runtime, like:

- Name of variables
- Unit or description of variables
- Position in the source code (file name and line number)

- Static text

A set of static data is assigned to a unique Message ID.」(SRS_Dlt_00024, SRS_Dlt_00027)

[SWS_Dlt_00129]「Static data shall not be transmitted in the Non-Verbose Mode.

Non-static data are all modifiable data, like values of variables. Only non-static data shall be transmitted within non-verbose mode.
The Non-Verbose Mode can be used in small ECU's with low memory and / or within a network with limited bandwidth. Because static data are not transmitted, the data also need not to be stored on the ECU's RAM/ROM.」()

7.7.5.1.1 Message ID

The Message ID is associated with additional information that contains all static data and allows the receiver to interpret all non-static data received. This additional information is provided externally (e.g. by an external file). Any number of data and any data type can be associated with a single Message ID. The receiver can interpret the data with the external description. For one Message ID there is only one unique description of the transmitted data and one combination of static data is assigned to one unique Message ID.

[SWS_Dlt_00329]「Message ID is a 32-bit unsigned integer.」()

[SWS_Dlt_00352]「Message ID shall be assigned unique for a single combination of static data.」(SRS_Dlt_00025, SRS_Dlt_00027)

[SWS_Dlt_00353]「With the combination of a Message ID and an external description, following information shall be recoverable that is otherwise provided in the Type Info:

- Type Length
- Data Type
- String Coding
- Variable Info
- Fixed Point」()

[SWS_Dlt_00134]「With the combination of a Message ID and an external description, following information shall be recoverable that is otherwise provided in the Extended Header:

- Message Type (MSTP)
- Message Info (MSIN)
- Number of arguments (NOAR)
- Application ID (APID)
- Context ID (CTID)」()

7.7.5.1.2 Assembly of Non-Static Data

This example will demonstrate how the non-static data is assembled, transmitted and interpreted.

Following information will be transmitted to an external client by the sending of a log message:

- static text: “Temperature measurement”
- 8-bit unsigned integer: measurement_point = 1 (no unit)
- 32-bit float: reading = 22.1 Kelvin

There is a unique Message ID that characterize this log message call on this specific position in the source code. Following information is associated with this Message ID:

- position in source code: source file “temp_meas.c”, line number 42
- static text: “Temperature measurement”
- expecting the value of a 8-bit unsigned integer with variable name = “measurement_point” and unit = “”
- expecting the value of a 32-bit float with variable name = “reading” and unit = “Kelvin”

All static data is already associated with the Message ID and only the non-static data will be transmitted:

Length in Bit	Value	Description
8	1	8-bit unsigned integer
32	22.1	32-bit float

Table 7-22 Assembly of non-static data in Non-Verbose Mode

Based on the Message ID, the receiver can reassemble all static data of this Dlt message (position in source code, static text, variable names and units). The non-static data will be transmitted consistently packed. The interpretation is possible by using the information associated with the Message ID. Also the ordering of the arguments is associated with the Message ID.

[SWS_Dlt_00378] ⌈The non-static data shall be transmitted consistently packed and byte aligned.⌋ (SRS_Dlt_00014, SRS_Dlt_00023)

7.7.5.1.3 Description Format for transmitted Data

An external file holds the information how the payload shall be interpreted. For describing transmitted messages which are in non verbose mode the ASAM Fibex (Field Bus Exchange Format) shall be used.

The software supplier of a SW-C or the BSW shall provide this description file. Because Dlt can have several sources of log or trace messages (several SW-Cs, Dem, Det) the provided description files can be merged to one file for a given ECU.

Each Fibex description file for describing Non Verbose messages only corresponds to log or trace messages for one ECU. This is because Message IDs are only unique per ECU. Additionally the Software Version Number of the ECU has to be provide by the description file.

[SWS_Dlt_00402] ⌈Each description file shall contain only one ECU XML-element.⌋()

[SWS_Dlt_00403] ⌈The ECU XML-element shall be extended by a SW_VERSION XML-element.

In principle each log or trace message is comparable to a PDU known in some network protocols. Here the description of a log or trace message shall be equivalent to a CAN-Frame specified by Fibex. The information from the Extended Header is put in additionally XML-elements inside the FRAME-TYPE XML-element. The Non-Static Data is described by PDU and SIGNAL XML-elements.

As seen from the user, a log or trace message has several arguments. These arguments can be static text or non static variables. Only the non static variables data is transmitted. To reassemble the whole message with all arguments, a FRAME XML-element shall contain some empty PDU XML-elements which represents arguments with static text. This text shall be placed in the DESC XML-element of the PDU XML-element.⌋()

[SWS_Dlt_00418] ⌈The ASAM Fibex Standard (Field Bus Exchange Format) Version 3.0 shall be used for describing a Non Verbose message.⌋(SRS_Dlt_00024, SRS_Dlt_00026)

[SWS_Dlt_00396] ⌈One log or trace message shall be represented by one FRAME XML-element in Fibex.⌋()

[SWS_Dlt_00397] ⌈The Message ID shall be the ID attribute of the <FRAME> XML-element.⌋()

[SWS_Dlt_00398] ⌈The <FRAME-TYPE> XML-element shall be extended by the following XML-elements:

- Message Type (MSTP) – MESSAGE_TYPE
- Message Info (MSIN) – MESSAGE_INFO
- Application ID (APID) – APPLICATION_ID
- Context ID (CTID) – CONTEXT_ID
- Source file – MESSAGE_SOURCE_FILE
- line number – MESSAGE_LINE_NUMBER⌋(SRS_Dlt_00026, SRS_Dlt_00027)

[SWS_Dlt_00399] ⌈The user data of the log or trace message shall be represented by several PDU XML-elements. Each argument shall get one PDU XML-element. ⌋ ()

[SWS_Dlt_00400] ⌈If the argument contains only static text, this text shall be placed in the DESC XML-element of the PDU. In this case the BYTE-LENGTH of the PDU XML-element shall be zero.⌋ (SRS_Dlt_00025)

[SWS_Dlt_00401] ⌈If the argument contains “Non-Static Data” the data transported in the message is described within the PDU as SIGNAL XML-element.⌋ (SRS_Dlt_00026)

The following example shows the description of a sample Dlt message in FIBEX XML.

```
<fx:FRAME ID="ID 1">
     <ho:SHORT-NAME>Dlt Message with ID 1</ho:SHORT-NAME>
     <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>
     <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE>
     <fx:PDU-INSTANCES>
          <fx:PDU-INSTANCE ID="P 1 0">
               <fx:PDU-REF ID-REF="PDU 1 0"/>
               <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P 1 1">
               <fx:PDU-REF ID-REF="PDU 1 1"/>
               <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P 1 2">
               <fx:PDU-REF ID-REF="PDU 1 2"/>
               <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
     </fx:PDU-INSTANCES>
     <fx:MANUFACTURER-EXTENSION>
          <MESSAGE TYPE>DLT TYPE LOG</MESSAGE TYPE>
          <MESSAGE INFO>DLT LOG DEBUG</MESSAGE INFO>
          <APPLICATION ID>APPI</APPLICATION ID>
          <CONTEXT ID>CONI</CONTEXT ID>
          <MESSAGE SOURCE FILE>demo.c</MESSAGE SOURCE FILE>
          <MESSAGE LINE NUMBER>72</MESSAGE LINE NUMBER>
     </fx:MANUFACTURER-EXTENSION>
</fx:FRAME>
<!--================ 1. Parameter ==================-->
<fx:PDU ID="PDU 1 0">
     <ho:SHORT-NAME></ho:SHORT-NAME>
     <ho:DESC>Temperature measurement</ho:DESC>
     <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
     <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
</fx:PDU>
<!--================ 2. Parameter ==================-->
<fx:PDU ID="PDU 1 1">
     <ho:SHORT-NAME>measurement point</ho:SHORT-NAME>
     <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>
     <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
```

```
      <fx:SIGNAL-INSTANCES>
        <fx:SIGNAL-INSTANCE ID="S 1 0">
           <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
           <fx:SIGNAL-REF ID-REF="S UINT8"/>
        </fx:SIGNAL-INSTANCE>
      </fx:SIGNAL-INSTANCES>
</fx:PDU>
<!--================ 3. Parameter ==================-->
<fx:PDU ID="PDU 1 2">
      <ho:SHORT-NAME>reading</ho:SHORT-NAME>
      <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
      <fx:SIGNAL-INSTANCES>
        <fx:SIGNAL-INSTANCE ID="S 1 0">
           <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
           <fx:SIGNAL-REF ID-REF="FLOA32"/>
        </fx:SIGNAL-INSTANCE>
      </fx:SIGNAL-INSTANCES>
</fx:PDU>
```

7.7.5.2 Verbose Mode

In contrary to the Non-Verbose Mode, the Verbose Mode provides all information for interpreting the transmitted data within the message. The data is self-describing. All static information like strings is transmitted. No extra description file is needed, because of transmitting all information within the log and trace message.

The Verbose Mode can be used on ECU's where enough memory and high network bandwidth are available. Because of the self-description, the stored data on the external client is interpretable at any time and without any further information.

7.7.5.2.1 Dlt Message Format in General

In Verbose Mode, any desired number of arguments can be transmitted. The information about the payload is provided within the message. The payload adjoins the Extended Header and consists of one or more arguments. The number of arguments in the payload is specified in the Extended Header in the field Number of arguments (NOAR).

Each argument consists of a "Type Info" field and the appended Data Payload. In "Type Info" field the necessary information is provided to interpret the following data structure.

[SWS_Dlt_00459] ⌈The assembly of a Dlt message in Verbose Mode is shown in the following table (Table 7-23) and shall be in this format.

Length (byte)	Name	Short Description
4,8,12 or 16	**Standard Header**	Essential information for interpreting the Dlt message

Length (byte)		Name	Short Description
10		**Extended Header**	Additional information about the Dlt message
		Payload - *list of arguments*	
		Argument 1	
	4	Type Info	Essential information for interpreting the Data Payload
	x	Data Payload	Data and optional additional parameters like variable info
		Argument n	
	4	Type Info	Essential information for interpreting the Data Payload
	x	Data Payload	Data and optional additional parameters like variable info
		End of list of arguments	

Table 7-23 Assembly of a Dlt Message in Verbose Mode

⌋(SRS_Dlt_00023, SRS_Dlt_00044)

[SWS_Dlt_00409] ⌈The arguments and all inherited data shall be transmitted consistently packed. ⌋(SRS_Dlt_00023)

7.7.5.2.2 Type Info

[SWS_Dlt_00421] ⌈The bit field Type Info shall be used, if the Verbose (VERB) bit is set AND if Message Type (MSTP) equals DLT_TYPE_LOG or DLT_TYPE_APP_TRACE.

With Type Info, the correct structure of the following Data is chosen automatically.⌋(SRS_Dlt_00044)

[SWS_Dlt_00135] ⌈Type Info is a bit field of 32 bit. Following Bit field shall be used for Type Info:

Bit position	Name	Description
Bit 0 - 3	Type Length (TYLE)	the length of the standard data 0x00 = not defined 0x01 = 8 bit 0x02 = 16 bit 0x03 = 32 bit 0x04 = 64 bit 0x05 = 128 bit 0x06 - 0x07 reserved for future use
Bit 4	Type Bool (BOOL)	is set if the data is a bool data

Bit position	Name	Description
Bit 5	Type Signed (SINT)	is set if the data is a signed integer data
Bit 6	Type Unsigned (UINT)	is set if the data is a unsigned integer data
Bit 7	Type Float (FLOA)	is set if the data is a float data
Bit 8	Type Array (ARAY)	is set if the data is an array of standard types
Bit 9	Type String (STRG)	is set if the data is a string
Bit 10	Type Raw (RAWD)	is set if the data is raw data
Bit 11	Variable Info (VARI)	is set if additional information to a variable (like name, unit) is given.
Bit 12	Fixed Point (FIXP)	is set if quantization and offset are added
Bit 13	Trace Info (TRAI)	is set if additional trace information is added (like module name / function name)
Bit 14	Type Struct (STRU)	is set if the data is a struct like specified in C (for future use)
Bit 15 – 17	String Coding (SCOD)	is the coding of the Type String (STRG) 0x00 = ASCII 0x01 = UTF-8 0x02 - 0x07 reserved for future use
Bit 18 – 31	reserved for future use	

Table 7-24 Assembly of Type Info

The table below shows a simplified assembly of Type Info

		Bit position							
Offset to start pos in byte	**Field Name**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	Type Info	TYLE	TYLE	TYLE	TYLE	BOOL	SINT	UINT	FLOA
1	Type Info	ARAY	STRG	RAWD	VARI	FIXP	TRAI	STRU	SCOD
2	Type Info	SCOD	SCOD	-	-	-	-	-	-
3	Type Info	-	-	-	-	-	-	-	-

Table 7-25 Simplified Assembly of Type Info

The entries of Type Info are specified in the following section in detail.⌋ (SRS_Dlt_00044)

Bits Type Length (TYLE)

Type Length specifies the length of the standard data type.

[SWS_Dlt_00354] ⌈Type Length is a bit field of 4 bit.

Type Info contains

- 1 (8 bit) for bool data (BOOL)
- 1 (8 bit) or 2 (16 bit) or 3 (32 bit) or 4 (64 bit) or 5 (128 bit) for signed (SINT) and unsigned integer data (UINT)
- 2 (16 bit) or 3 (32 bit) or 4 (64 bit) or 5 (128 bit) for float data (FLOA) ⌋()

Bit Variable Info (VARI)

If Variable Info (VARI) is set, the name and the unit of a variable can be added. Both always contain a length information field and a field with the text (of name or unit). The length field contains the number of characters of the associated name or unit filed. The unit information is to add only in some data types.

[SWS_Dlt_00410] ⌈The coding of all text in Variable Info (VARI) shall be in 8-bit ASCII format. ⌋()

[SWS_Dlt_00411] ⌈The strings in VARI shall be null terminated. ⌋()

Bit Fixed Point (FIXP)

If fixed point values are used, the Fixed Point (FIXP) bit shall be set. Than the Data field represents the physical value of a fixed point variable.
For interpreting the fixed point variable the logical value of this variable has to be calculated.
The logical value is calculated by the physical value, the quantization and the offset of fixed point variable.

[SWS_Dlt_00389] ⌈The following equation defines the relation between the logical value (log_v) and the physical value (phy_v), offset and quantization:
log_v = phy_v * quantization + offset ⌋()

[SWS_Dlt_00169] ⌈The bit Fixed Point (FIXP) shall only be set in combination with Type Signed (SINT) or Type Unsigned (UINT). ⌋()

Bits String Coding (SCOD)

String Coding specifies only the coding of string data of Type String (STRG). All other strings like parameter name, unit and description are coded in 8-bit ASCII format.

[SWS_Dlt_00182**]** ⌈String Coding is a bit field of 3 bit. ⌋()

[SWS_Dlt_00366] ⌈Following values shall be used for String Coding (SCOD):
0x00 = ASCII
0x01 = UTF-8

0x02 - 0x07 reserved for future use ⌋()

[SWS_Dlt_00183] ⌈String Coding shall be set if Type String (STRG) is set. ⌋()

[SWS_Dlt_00367] ⌈String Coding shall be set if Trace Info (TRAI) is set. ⌋()

7.7.5.2.3 Data Payload

Type Bool (BOOL)

[SWS_Dlt_00422] ⌈If the BOOL bit is set, the Data Payload shall consist of at least one 8-bit unsigned integer parameter. ⌋()

[SWS_Dlt_00423] ⌈If the Data field equals 0x0, it shall be interpreted as FALSE. If the Data field equals 0x1 it shall be interpreted as TRUE. ⌋()

[SWS_Dlt_00139] ⌈Type Length (TYLE) shall be 1. ⌋()

[SWS_Dlt_00355] ⌈If Variable Info (VARI) is set, the Length of Name, the Name and the Unit fields shall be added. ⌋()

[SWS_Dlt_00369] ⌈The Data Payload of Type Bool (BOOL) shall be assembled as shown in following table.

Length in bit		**Name**	**Description**
If **Variable Info (VARI)** is set in Type Info			
	16	Length of Name + termination char.	Unsigned 16-bit integer
	x	Name	Null terminated string (name of variable)
8		**Data**	0x0 if value is FALSE or 0x1 if value is TRUE

Table 7-26 Data Payload of Type Bool (BOOL)

⌋()

Type Signed (SINT) and *Type* Unsigned (UINT)

The SINT and UINT Data Payload are assembled in the same way. The only difference is in interpreting the Data field.

[SWS_Dlt_00385] ⌈If the SINT bit is set, the Data Payload consists of at least one signed integer Data field. ⌋ ()

[SWS_Dlt_00386] ⌈If the UINT bit is set, the Data Payload consists of at least one unsigned integer Data field.

Variable Info (VARI) and Fixed Point (FIXP) are optional.

Length in bit		Name	Description
If **Variable Info (VARI)** is set in Type Info			
	16	Length of Name + termination char.	Unsigned 16-bit integer
	16	Length of Unit + termination char.	Unsigned 16-bit integer
	x	Name	Null terminated string (name of variable)
	x	Unit	Null terminated string (unit of variable)
If **Fixed Point (FIXP)** is set in Type Info			
	32	Quantization	32-bit float
	32 / 64 / 128	Offset	Signed integer - with the length of at least 32 bit. The length shall be: 32 bit if Type Length (TYLE) equals 1,2 or 3 64 bit if Type Length (TYLE) equals 4 or 128 bit if Type Length (TYLE) equals 5
8/16/32 / 64 / 128		**Data**	Length depends on TYLE

Table 7-27 Data Payload of Type Signed (SINT) and Type Unsigned (UINT)

⌋ ()

[SWS_Dlt_00356] ⌈Type Length (TYLE) shall be set to 1, 2, 3, 4 or 5. ⌋ ()

[SWS_Dlt_00357] ⌈If Variable Info (VARI) is set, the "Length of Name", "Length of Unit", the "Name" and the "Unit" fields shall be added. ⌋ ()

[SWS_Dlt_00412] ⌈If FIXP is set, the Quantization and Offset fields shall be added. ⌋ ()

[SWS_Dlt_00388] ⌈The Quantization field shall be a 32-bit float field. ⌋ ()

[SWS_Dlt_00387] ⌈The Offset field is a singed integer field with at least 32 bit. If the TYLE equals 4 the Offset field shall be a 64 singed integer field and if the TYLE equals 5 the Offset field shall be a 128 singed integer field. ⌋ ()

[SWS_Dlt_00358] ⌈The length of Data shall depend on Type Length (TYLE). ⌋ ()

[SWS_Dlt_00370] ⌈The Data Payload of Type Signed (SIGN) and of Type Unsigned (UINT) shall be assembled as shown in Table 7-27. ⌋()

Type Float (FLOA)

[SWS_Dlt_00390] ⌈If the bit Type Float (FLOA) is set, the Data Payload shall consist of at least one Data field, which shall be interpreted as a float variable.

Variable Info (VARI) is optional.

Length in bit		Name	Description
If **Variable Info (VARI)** is set in Type Info			
	16	Length of name + termination char.	Unsigned 16-bit integer
	16	Length of unit + termination char.	Unsigned 16-bit integer
	x	Name	Null terminated string (name of variable)
	x	Unit	Null terminated string (unit of variable)
8/16/32 / 64 / 128		**Data**	Float data length depends on TYLE

Table 7-28 Data Payload of Type Float (FLOA)

⌋()

[SWS_Dlt_00145] ⌈Type Length (TYLE) shall be set to 2, 3, 4 or 5 as specified in IEEE 754r:

Type Length (TYLE)	Type	Length	Mantissa	Exponent
2	b16 bit	16 bit	10 bit	5
3	b32 bit (single)	32 bit	23 bit	8
4	b64 bit (double)	64 bit	52 bit	11
5	b128	128 bit	112 bit	15

Table 7-29 Definition of Type Length according to IEEE 754r

⌋()

[SWS_Dlt_00362] ⌈If Variable Info (VARI) is set, the “Length of Name”, “Length of Unit”, the “Name” and the “Unit” fields shall be added. ⌋()

[SWS_Dlt_00363] ⌈The length of Data shall depend on Type Length (TYLE). ⌋()

[SWS_Dlt_00371] ⌈The argument of Type Float (FLOA) shall be assembled as shown in Table 7-28.⌋()

***Type* String (STRG)**

[SWS_Dlt_00420] ⌈If the bit Type String (STRG) is set, the Data Payload shall consist of at least one Data field, which shall be interpreted as a string variable.⌋ (SRS_Dlt_00025)

[SWS_Dlt_00155] ⌈String Coding (SCOD) shall be specified.⌋()

[SWS_Dlt_00392] ⌈The string in the Data field shall be interpreted with the type corresponding to the String Coding (SCOD) field in the Type Info field.⌋()

[SWS_Dlt_00156] ⌈At the beginning of the Data Payload, a 16-bit unsigned integer specifies the length of the string (provide in the Data field) in byte including the termination character.⌋()

[SWS_Dlt_00157] ⌈If Variable Info (VARI) is set, the “Length of Name” and the “Name” fields shall be added.⌋()

[SWS_Dlt_00373] ⌈The Data Payload of Type String (STRG) shall be assembled as shown in following table.

Length in bit		**Name**	**Description**
16		Length of string + termination char.	Unsigned 16-bit integer
If **Variable Info (VARI)** is set in Type Info			
	16	Length of name + termination char.	Unsigned 16-bit integer
	x	Name	Null terminated string (name of variable)
x		**Data** string	Null terminated data string

Table 7-30 Data Payload of Type String (STRG)

⌋()

Type Array (ARAY)

[SWS_Dlt_00147] ⌈If the bit Type Array is set, the Data Payload shall consist of an n-dimensional array of one or more data types of bool (BOOL), signed integer

(SINT), unsigned integer (UINT) or float (FLOA) data types. The TYLE field and FIXP field shall be interpreted as in the standard data types. ⌋()

[SWS_Dlt_00148] ⌈At the beginning of the Data Payload a 16-bit unsigned integer shall specify the number of dimensions of the array. ⌋()

[SWS_Dlt_00149] ⌈If Variable Info (VARI) is set, the name of the array shall be described. ⌋()

[SWS_Dlt_00150] ⌈Within the loop over the number of dimensions, a 16-bit unsigned integer shall specify the number of entries in the current dimension. ⌋()

[SWS_Dlt_00152] ⌈If Variable Info (VARI) is set, the "Length of Name", "Length of Unit", the "Name" and the "Unit" fields shall be added. ⌋()

[SWS_Dlt_00153] ⌈If Fixed Point (FIXP) bit is set in the Type Info, the quantization and offset for the entry in the array shall be added.

It is only possible to use the same fixed point calculation for all entries in the array. ⌋()

[SWS_DIt_00372] ⌈The Data Payload of Type Array (ARAY) shall be assembled as shown in following table.

Length in bit		Name	Description
16		Number of dimensions	Unsigned 16-bit integer
Loop over number of dimensions			
	16	Number of entries in current dimension	Unsigned 16-bit integer
Loop *End*			
If **Variable Info (VARI)** is set in Type Info of current dimension			
	16	Length of Name + termination char.	Unsigned 16-bit integer
	16	Length of Unit + termination char.	Unsigned 16-bit integer
	x	Name	Null terminated string (name of current dimension)
	x	Unit	Null terminated string (unit of current dimension)
If **Fixed Point (FIXP)** is set in Type Info of current dimension			
	32	Quantization	32-bit float
	32 / 64 / 128	Offset	Signed integer of 32 bit if Type Length (TYLE) <= 3 or 64 bit if Type Length (TYLE) = 4 or 128 bit if Type Length (TYLE) = 5
x		**Data** of whole array The data shall be in the same structure/ordering as it is defined in the C90 standard [ii]	

Table 7-31 Data Payload of Type Array (ARAY)

⌋()

***Type* Struct (STRU)**

If this bit is set, structured data are transmitted.

[SWS_DIt_00175] ⌈At the beginning of the Data Payload a 16-bit unsigned integer shall specify the number of entries of the structure or the object. ⌋()

[SWS_DIt_00176] ⌈If Variable Info (VARI) is set, the “Length of Name” and the “Name” fields shall be added. ⌋()

[SWS_DIt_00177] ⌈The list of entries contains one or more standard arguments with Type Info and Data Payload. All standard argument types are allowed. ⌋()

[SWS_Dlt_00414] ⌈The Data Payload of Type Struct (STRU) shall be assembled as shown in following table.

Length (bit)		Name	Description
16		Number of entries in the struct / object	Unsigned 16-bit integer
If **Variable Info (VARI)** is set in Type Info			
	16	Length of name + termination char.	Unsigned 16-bit integer
	x	Name	Null terminated string (name of variable)
List of entries (each entry consists of a standard argument type described above)			
		Entry 1	
	4	Type Info	Essential information for interpreting the Data Payload
	x	Data Payload	Data and optional additional parameters like variable info
		Entry n	
	4	Type Info	Essential information for interpreting the Data Payload
	x	Data Payload	Data and optional additional parameters like variable info
End of list of entries			

Table 7-32 Data Payload of Type Struct (STRU)

⌋()

***Type* Raw (RAWD)**

If this bit is set, the Data Payload describes raw data. Variable Info (VARI) is optional.

[SWS_Dlt_00364] [If Variable Info (VARI) is set, the coding of the name shall be in 8-bit ASCII format. ⌋()

[SWS_Dlt_00160] ⌈At the beginning of the Data Payload a 16-bit unsigned integer shall specify the length of the raw data in byte. ⌋()

[SWS_Dlt_00161] ⌈If Variable Info (VARI) is set, the “Length of Name” and the “Name” fields shall be added.

The interpretation of the Data field in the case of a Raw argument can not be done. Some tools can show this data by a user defined data type. ⌋()

[SWS_Dlt_00374] ⌈The Data Payload of Type Raw (RAWD) shall be assembled as shown in following table.

Length in bit		Name	Description
16		Length of raw data in byte	Unsigned 16-bit integer
If **Variable Info (VARI)** is set in Type Info			
	16	Length of Name + termination char.	Unsigned 16-bit integer
	x	Name	Null terminated string (description of variable)
x		**Data**	Raw data

Table 7-33 Data Payload of Type Raw (RAWD)

⌋()

***Type* Trace Info (TRAI)**

Trace info is a separate argument in the Dlt message.

[SWS_Dlt_00170] ⌈If the bit Trace Info (TRAI) is set, the trace information (like module name / function) shall be transmitted in the argument. ⌋()

[SWS_Dlt_00172] ⌈At the beginning of the Data Payload, a 16-bit unsigned integer shall specify the length of the trace data string in byte including the termination character. ⌋()

[SWS_Dlt_00173] ⌈Null terminated trace data string shall follow. ⌋()

[SWS_Dlt_00171] ⌈String Coding (SCOD) shall specify the coding of the trace data string. ⌋()

[SWS_Dlt_00375] ⌈The Data Payload of Trace Info (TRAI) shall be assembled as shown in following table.

Length in bit	Name	Description
16	Length of string + termination char. (in byte)	Unsigned 16-bit integer
x	**Trace Data String**	Null terminated string (like name of module / function in packet)

Table 7-34 Data Payload of Trace Info (TRAI)

⌋()

7.7.5.2.4 Example of representation of natural data type argument

The following example shows the assembly of an 8-bit unsigned integer argument with Variable Info (VARI) bit set in verbose mode.

The Type Info is a 32-bit field that describes the Data. In this example, it defines the variable type (unsigned integer), its length (8 bit) and the presence of Variable Info (VARI) that describes the name and unit of the variable. Variable Info is following with two 16-bit unsigned integers describing the length of the Name and the Unit of the variable. Two null terminated strings follow that describe the Name and the Unit. Finally, the variable value follows. The length of the Data field is 8 bit.

Length in bit		**Name**	**Value**	**Description**
32		**Type Info**	0001 0010 0001 0000 0000 0000 0000 0000	Type Length (TYLE) = 0x1 (8 bit) Type Unsigned (UINT) = 0x1 Variable Info (VARI) = 0x1
Variable Info (VARI) is set in Type Info				
	16	Length of name + termination char.	12	Unsigned 16-bit integer
	16	Length of unit + termination char.	8	Unsigned 16-bit integer
	96 (12*8)	Name	temperature	Null terminated string (name of variable)
	64 (8*8)	Unit	celsius	Null terminated string (unit of variable)
8		**Data**	25	

Table 7-35 Example of the assembly of the payload in verbose mode

List of different Type Info fields bits combinations

The following table shows all combinations of valid settings in Type Info sorted according to the bit position in Type Info.

0-3 TYLE				4 BOOL	5 SINT	6 UINT	7 FLOA	8 ARAY	9 STRG	10 RAWD	11 VARI	12 FIXP	13 TRAI	14 STRU	15-17 SCOD			18-31 RESERVED	
x	x	x	x	x							o								
x	x	x	x		x						o	o							
x	x	x	x			x					o	o							
x	x	x	x				x				o								
									x		o				x	x	x		
										x	o								
													x		x	x	x		
								x			o								

0-3 TYLE				4 BOOL	5 SINT	6 UINT	7 FLOA	8 ARAY	9 STRG	10 RAWD	11 VARI	12 FIXP	13 TRAI	14 STRU	15-17 SCOD			18-31 RESERVED	
											o			x					
											o								

Table 7-36 Assembly of valid settings in Type Info (o – optional, x – mandatory for this type, empty – not allowed for this type)

The following table shows the mandatory (marked with x) and optional (marked with o) setting according to used variable type:

Valid Settings / Variable Type	Type Length (TYLE)	Variable Info (VARI)	Fixed Point (FIXP)	String Coding (SCOD)
Type Bool (BOOL)	x	o		
Type Signed Integer (SINT)	x	o	o	
Type Unsigned Integer (UINT)	x	o	o	
Type Float (FLOA)	x	o		
Type Array (ARAY)		o		
Type String (STRG)		o		x
Type Raw (RAWD)		o		
Trace Info (TRAI)				x
Type Struct (STRU)		o		

Table 7-37 Assembly of valid settings in Type Info (o – optional, x – mandatory for this type, empty – not allowed for this type)

Using the Verbose Mode helps to understand, analyze and debug the SW-C.

7.7.5.2.5 Recommended arguments

To identify the source of a log or trace message some information to find the location in the source code shall be added to a Dlt message. Therefore the first two arguments in a Dlt message shall be:

- the name of the source file (string) and
- the line number (unsigned integer).

[SWS_Dlt_00424] ⌈The first argument of a log or trace message shall be a string argument where the field “Name” (in Variable Info) contains the string “source_file” and the Data field contains the URL to the source file. ⌋ ()

[SWS_Dlt_00425] ⌈The second argument of a log or trace message shall be a UINT argument (with 32 bit) where the field “Name” (in Variable Info) contains the string “line_number” and the Data field contains the line number in the source file where the log or trace message is sent. ⌋()

7.7.6 Additional Message Parts

7.7.6.1 Extensions for storing in a database/file

The Dlt module can leave out some information in the header like timestamp and ECU ID. Therefore, it is important to store some additional information by the receiving external client.

For additionally storing the timestamp and the ECU ID a Storage Header shall be added in front of every received Dlt message.
Timestamp and ECU ID can be left of Dlt side, because of that the receiver shall add this information at receiving time. The Timestamp is also for a better calculating of sequences and timely dependencies by a diagnostic and visualization tool.
Additionally at the beginning of the Storage header a pattern shall be attached. This pattern is for some error recoveries if the byte-stream or file is broken.

[SWS_Dlt_00405] ⌈An external client shall add the Storage Header to a received Dlt message before it stores the message.

Offset	Length (byte)	Name	Description
	Dlt log or trace storage extension		
0	4	DLT-Pattern	“DLT”+0x01 in Hex 0 x 44 4C 54 01
4	8	Timestamp	
	4	seconds	Unsigned integer 32 bit seconds since 01.01.1970 (unix time)
	4	microseconds	Singed integer 32 bit microseconds of the second (between 0 – 1.000.000)
12	4	ECU ID	Four characters the ECU ID
16	**Dlt log or trace message**		
		Header	
		Extended Header	
		Payload	

Table 7-38 Storage Header to store in front of a Dlt message.

⌋()

[SWS_Dlt_00427] ⌈The first entry in the Storage Header shall be a pattern 0 x 44 4C 54 01 (“DLT”+0x1). ⌋()

[SWS_Dlt_00404] ⌈If an external client receives a message it shall store the time when it receives the message additionally to the message in the storage header. ⌋()

[SWS_Dlt_00292] ⌈If an external client receives a message it shall store the ECU ID when it receives the message additionally to the message in the storage header. ⌋()

[SWS_Dlt_00415] ⌈The first message stored in the database/file shall be the response to the “Get ECU Software Version” control message. ⌋()

7.7.7 Predefined messages

To control the internal behavior of the Dlt module it is important to have an interface to make some changes on the configuration of the Dlt module.

[SWS_Dlt_00186] ⌈The Dlt module shall handle the communication with an external client described in this chapter. ⌋()

7.7.7.1 Control messages

Control messages do not contain log or trace information. The use of control messages is to configure the internal behavior of the Dlt module and to get information of registered Application IDs and Context IDs.

NOTE: Control messages can not be send by SW-Cs or BSW modules. They are only exchanged between Dlt and an external client.

[SWS_Dlt_00187] ⌈Control messages are in general normal Dlt messages with a Standard Header, an Extended Header and payload.

Length in Byte	**Elements of the message**	**Description**
4,8,12 or 16	**Standard Header**	ECU equals receiver ECU name
10	**Extended Header**	MSTP equals DLT_TYPE_CONTROL MSCI equals DLT_CONTROL_REQUEST or DLT_CONTROL_RESPONSE
	Payload	
4	**Service_ID**	The service to execute

x	**Parameters**	Data for executing the service

Table 7-39 Overview of the elements of a control message

⌋(SRS_Dlt_00032)

[SWS_Dlt_00188] ⌈In the Standard Header the Application ID, Context ID, Session ID and Timestamp may be left blank. This means that they should be filled with zeros (0) if it not used otherwise in the control message. The fields Timestamp and SessionID of the Standard Header can be left by setting the bits WSID or WTMS to zero.
All other fields have the same meaning as in a log message, except for the ECU field. If the external client is sending a message to a given ECU, it shall insert the ECU ID of the receiver. ⌋()

[SWS_Dlt_00189] ⌈Control messages shall be sent in Non-Verbose Mode. The description of the transmitted data comes from this specification. ⌋()

[SWS_Dlt_00190] ⌈If a specific functionality like switch to Verbose Mode or message filtering is not implemented in Dlt it shall response with NOT_SUPPORTED. ⌋()

[SWS_Dlt_00416] ⌈If the provided parameters of a request are not correct, Dlt shall answer with status ERROR. ⌋()

[SWS_Dlt_00417] ⌈If a new request is received by Dlt while an old one is not finished it shall answer to the new one with ERROR. ⌋()

[SWS_Dlt_00191] ⌈The payload shall always begin with the Service_ID. This ID is for identifying the request and the corresponding action to be processed by the Dlt module. ⌋()

[SWS_Dlt_00192] ⌈The communication procedure between external client and Dlt module on the ECU is like following:

1. The external client sends a request to the Dlt module (MSCI == DLT_CONTROL_REQUEST)
2. The Dlt module receives this request and performs the requested action
3. The Dlt module sends a response (MSCI == DLT_CONTROL_RESPONSE), with the same Service_ID of the request back to the external client. ⌋()

[SWS_Dlt_00193] ⌈The external client shall not send any new requests until the response of the Dlt module is received. The procedure is synchronous and not reentrant.

If the external client did not receive an answer for a control message from Dlt it can repeat the control message after a timeout. The length of the timeout is chosen by the external client, depending on the performance of the ECU and on the communication channel bandwidth.

The following tables describe the request and response messages for a specific Service_ID. The Service_ID and the request or response parameter represent the payload. The Number column is the ordering of the parameters. ⌋()

7.7.7.1.1 Set Log Level

[SWS_Dlt_00194] ⌈

Service name:	Set_LogLevel		
Service_ID [hex]	0x01		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***Application_ ID***	4 * uint8	Representation of the Application ID. If this field is filled with NULL all log level for all Context IDs on this ECU are set.
2	***Context_ID***	4 * uint8	Representation of the Context ID • If this field is filled with NULL all Context IDs belonging to the given Application ID are set. • is only interpreted if **Application ID** is not NULL
3	***new_log_level***	sint8	the new log level to set • can be in the range of DLT_LOG_FATAL to DLT_LOG_VERBOSE for setting the pass through range • if set to 0 all messages from this Context ID are blocked • if set to -1 the default log level for this ECU will be used
4	***com_interface***	4*uint8	This field is used if Dlt supports filtering messaged for different interfaces differently. Than the log level for this interface can be provided. Otherwise it should be filled with zeros. To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to modify the pass through range for log messages for a given Context ID.		

⌋(SRS_Dlt_00032)

[SWS_Dlt_00195] ⌈Action to process:
Modify the table with Application IDs and Context IDs (see 7.6.4).⌋ (SRS_Dlt_00032)

7.7.7.1.2 Set Default Log Level

[SWS_Dlt_00380] ⌈

Service name:	Set_DefaultLogLevel		
Service_ID [hex]	0x11		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_log_level***	sint8	the new log level to set • can be in the range of DLT_LOG_FATAL to DLT_LOG_VERBOSE for setting the pass through range • if set to 0 all messages are blocked •
2	***com_interface***	4*uint8	This field is used if Dlt supports filtering messaged for different interfaces differently. Than the log level for this interface can be provided. Otherwise it should be filled with zeros. To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to modify the pass through range for log messages for all not explicit set Context IDs.		

⌋ (SRS_Dlt_00032)

[SWS_Dlt_00381] ⌈Action to process:
Modify the DltDefaultMaxLogLevel. ⌋ (SRS_Dlt_00032)

7.7.7.1.3 Set Trace Status

[SWS_Dlt_00196] ⌈

Service name:	Set_TraceStatus
Service ID [hex]	0x02
:	

Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***Application_ID***	`4 * uint8`	Representation of the Application ID. • If this field is filled with NULL all trace status for all Context IDs on this ECU are set.
2	***Context_ID***	`4 * uint8`	Representation of the Context ID • If this field is filled with NULL all Context IDs belonging to the given Application ID are set. • is only interpreted if **Application ID** is not NULL
3	***new_trace_status***	`sint8`	the new trace status to set • can be 1 – for On and 0 – for Off • if set to -1 the default trace status for this ECU will be used
4	***com_interface***	`4*uint8`	This field is used if Dlt supports filtering messaged for different interfaces differently. Than the log level for this interface can be provided. Otherwise it should be filled with zeros. To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to enable or disable trace messages for a given Context ID.		

⌋(SRS_Dlt_00032)

7.7.7.1.4 Set Default Trace Status

[SWS_Dlt_00383] ⌈

Service name:	Set_DefaultTraceStatus		
Service_ID [hex]	`0x12`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_trace_status***	`sint8`	the new trace status to set • can be 1 – for On and 0 – for Off

2	***com_interface***	`4*uint8`	This field is used if Dlt supports filtering messaged for different interfaces differently. Than the log level for this interface can be provided. Otherwise it should be filled with zeros. To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
Response Parameter			
Number	***Name***	**`Type`**	**Description**
1	***Status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to enable or disable trace messages for all not explicit set Context IDs		

⌋(SRS_Dlt_00032)

[SWS_Dlt_00382] ⌈Action to process:
Modify the default trace status. ⌋()

7.7.7.1.5 Get Log Info

[SWS_Dlt_00197] ⌈

Service name:	Get_LogInfo		
Service ID [hex]	`0x03`		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**

1	***Options***	`uint8`	• **1 -** reserved • **2 -** reserved • **3** - means only information about registered Application IDs and Context IDs without log level or trace status information • **4** - means information about registered Application IDs and Context IDs with log level and without trace status information • **5** - means information about registered Application IDs and Context IDs without log level and with trace status information • **6** - means information about registered Application IDs and Context IDs with log level and with trace status information • **7** - means information about registered Application IDs and Context IDs with log level and with trace status information and all textual descriptions of each Application ID and Context ID (If DltImplementVerboseMode is not set NOT_SUPPORTED shall be the response)
2	***Application_ID***	`4 * uint8`	Representation of the Application ID. • If this field is filled with NULL all Application IDs with all Context IDs registered with this ECU are requested
3	***Context_ID***	`4 * uint8`	Representation of the Context ID • If this field is filled with NULL all Context IDs belonging to the given Application ID are requested • is only interpreted if **Application ID** is not NULL
4	***com_interface***	`4*uint8`	This field is used if Dlt supports filtering messaged for different interfaces differently. Than the log level for this interface can be provided. Otherwise it should be filled with zeros. To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
Response Parameter			
Number	***Name***	**`Type`**	**Description**

1	***Status***	`uint8`	• **1** - NOT_SUPPORTED • **2** - ERROR • **3** - means only information about registered Application IDs and Context IDs without log level or trace status information • **4** - means information about registered Application IDs and Context IDs with log level and without trace status information • **5** - means information about registered Application IDs and Context IDs without log level and with trace status information • **6** - means information about registered Application IDs and Context IDs with log level and with trace status information • **7** - means information about registered Application IDs and Context IDs with log level and with trace status information and all textual descriptions of each Application ID and Context ID **NOTE:** In this case the control message shall be in Verbose Mode • **8** – NO matching Context IDs • **9** – RESPONSE DATA OVERFLOW – If the generated response is to large for transmitting a single Dlt message (PDU is 65536 Byte) only Status type RESPONSE DATA OVERFLOW shall be send (No Application_IDs and com_interface entries are transmitted). Than the user could than decide to formulate a less complex request. If the response is not of the Status 1, 2, 8 or 9 it should be the same that is used in the request entry of “Options”.
2	***Application_IDs***	`LogInfoType`	Null if Status == 1 or 2 Response is build like this: 1. Number of Application IDs 2. Application ID + Number of Context IDs 1. Context ID + log level; 2. Context ID + log level; 3. … 3. Application ID + Number of Context IDs 1. Context ID + log level; 2. 4. …l For a detailed description see 7.7.7.1.5.1 to 7.7.7.1.5.3
3	***com_interface***	`4*uint8`	This field is used if Dlt supports filtering messaged for different interfaces differently. Than the log level for this interface can be provided. Otherwise it should be filled with zeros. To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
Description:	Called to request information about registered Application IDs, there Context IDs and the corresponding log level. If DltImplementAppIdContextIdQuery is not set this shall response NOT_SUPPORTED.		

⌋(SRS_Dlt_00032, SRS_Dlt_00033)

7.7.7.1.5.1 LogInfoType

Name:	`LogInfoType`	
Type:	`structure`	
Type field:		
	`uint16 count_app_ids`	
	`AppIDsType[] app_ids`	
Description:		

7.7.7.1.5.2 AppIDsType

Name:	`AppIDsType`	
Type:	`structure`	
Type field:		
	`uint32 app_id`	
	`uint16 count_context_ids`	
	`ContexIDsInfoType[] context_id_info`	
optional, if options ==7	`uint16 len_app_description`	
	`String app_description`	
Description:	Holds information about a specific Application ID	

7.7.7.1.5.3 ContextIDsInfoType

Name:	`ContextIDsInfoType`	
Type:	`structure`	
Type field:		
	`uint32 context_id`	
if options == 4,6 or 7	`sint8 log_level`	
if options == 5,6 or 7	`sint8 trace_status`	
optional, if options == 7	`uint16 len_context_description`	
	`String context_description`	
Description:	Holds information about a specific Context ID	

7.7.7.1.6 Get Default Log Level

[SWS_Dlt_00198] ⌈

Service name:	Get_DefaultLogLevel		
Service ID [hex]	`0x04`		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
none			

Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***log_level***	uint8	Actual log level
Description:	Returns the actual default log level		

⌋(SRS_Dlt_00032)

7.7.7.1.7 Get Default Trace Status

[SWS_Dlt_00494] ⌈

Service name:	Get_DefaultTraceStatus		
Service ID [hex]	0x15		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***trace_status***	uint8	Actual Trace Status 0 - off, 1 - on
Description:	Returns the actual default trace status		

⌋()

7.7.7.1.8 Store Configuration

[SWS_Dlt_00199] ⌈

Service name:	Store_Config		
Service ID [hex]	0x05		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to store the actual configuration of Dlt to NVRAM (see 7.6.6)		

⌋()

7.7.7.1.9 Reset to Factory Default

[SWS_Dlt_00200] ⌈

Service name:	ResetToFactoryDefault		
Service ID [hex]	0x06		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to set the configuration of Dlt to factory defaults (see 7.6.6)		

⌋ ()

7.7.7.1.10 Set Communication Interface Status

[SWS_Dlt_00201] ⌈

Service name:	SetComInterfaceStatus		
Service ID [hex]	0x07		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***com_interface***	4*uint8	To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
2	***new_status***	uin8	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to enable or disable a specific communication interface.		

⌋ ()

7.7.7.1.11 Get Communication Interface Status

[SWS_Dlt_00501] ⌈

Service name:	GetComInterfaceStatus		
Service ID [hex]	0x16		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***com_interface***	4*uint8	To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0” Not used bytes are filled by zero.
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR 3 == INTERFACE NOT AVAILABLE
2	***if_status***	uin8	Current interface status 0 – OFF 1 – ON
Description:	Called to get the status information of a specific communication interface.		

⌋ ()

7.7.7.1.12 Get Communication Interface Names

[SWS_Dlt_00502] ⌈

Service name:	GetComInterfaceNames		
Service ID [hex]	0x17		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***count_if***	uin8	Count of transmitted interface names
3	***com_if_names***	4*uin8[]	List of communication interface names. Array on each 4 byte
Description:	Called to get all available communication interfaces.		

⌋ ()

7.7.7.1.13 Set Communication Maximum Bandwidth

[SWS_Dlt_00202] ⌈

Service name:	SetComInterfaceMaxBandwidth		
Service ID [hex]	0x08		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***com_interface***	4*uint8	To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0”
2	***max_bandwidth***	uint32	the maximum bandwidth to allow for this interface in **bit per second**
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to set the maximum bandwidth for a specific communication interface.		

⌋ (SRS_Dlt_00030)

7.7.7.1.14 Get Communication Maximum Bandwidth

[SWS_Dlt_00503] ⌈

Service name:	GetComInterfaceMaxBandwidth		
Service ID [hex]	0x18		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***com_interface***	4*uint8	To interpret as a name for a interface Possible values are “DCM” – Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: “SER1”, “eth0”
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR 3 == INTERFACE NOT AVAILABLE
2	***max_bandwidth***	uint32	the maximum bandwidth allowed for this interface in **bit per second**
Description:	Called to get the current maximum bandwidth for a specific communication		

	interface.

⌋()

7.7.7.1.15 Switch to Verbose Mode

[SWS_Dlt_00489] ⌈In the case of a SetVerboseMode or GetVerboseMode message, the Application ID (APID), Context ID (CTID) and the Session ID (SEID) may be filled in the header and extended header. If they are non zero the pair of APID and CTID together with the SEID (see Session ID in 7.6.4) identifies a unique client server interface of a SW-C/runnable which is called in respect to reception of this message to change the VerboseMode. If this fields are filled with zeros or are left (in case of SEID) all ContextIDs of the ECU shall be informed about the change of the Verbose mode by calling it's Dlt_SetVerboseMode_<SESSION> functions. ⌋()

[SWS_Dlt_00203] ⌈

Service name:	SetVerboseMode		
Service ID [hex]	0x09		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_status***	uint8	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to switch on/off the Verbose Mode. This works only if the Dlt module and all SW-Cs are compiled with Verbose Mode enabled. If DltImplementVerboseMode is not set the response shall be NOT_SUPPORTED.		

⌋()

[SWS_Dlt_00204] ⌈This service modifies the DltUseVerboseMode parameter. ⌋()

[SWS_Dlt_00504] ⌈

Service name:	GetVerboseModeStatus		
Service ID [hex]	0x19		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			

Number	Name	Type	Description
1	Status	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	mode_status	uint8	0 – OFF 1 – ON
Description:	Called to get f the Verbose Mode. This functionality is optionally, only if the Dlt module can track the verbose mode changes of the SW-C.		

⌋()

7.7.7.1.16 Filter messages

[SWS_Dlt_00205] ⌈

Service name:	SetMessageFilterering		
Service ID [hex]	0x0A		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	Name	Type	Description
1	new_status	uint8	0 – OFF 1 – ON
Response Parameter			
Number	Name	Type	Description
1	Status	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to switch on/off the message filtering by the Dlt module. If DltImplementFilterMessages is not set NOT_SUPPORTED shall be the response.		

⌋()

[SWS_Dlt_00206] ⌈This message modifies DltFilterMessages. ⌋()

[SWS_Dlt_00505] ⌈

Service name:	GetMessageFiltereringStatus		
Service ID [hex]	0x1A		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	Name	Type	Description
none			
Response Parameter			
Number	Name	Type	Description
1	Status	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
1	filter_status	uint8	0 – OFF 1 – ON

Description:	Called to get the message filtering status from the Dlt module.

⌋()

7.7.7.1.17 Set Timing Packets

[SWS_Dlt_00207] ⌈

Service name:	SetTimingPackets		
Service ID [hex]	0x0B		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_status***	uint8	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to switch on/off the continuous cyclic sending of timing packets. If DltImplementTimestamp is not set NOT_SUPPORTED shall be the response.		

⌋(SRS_Dlt_00028)

7.7.7.1.18 Get Local Time

[SWS_Dlt_00208] ⌈

Service name:	GetLocalTime		
Service ID [hex]	0x0C		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to get a single packet with timestamp (TMSP). The TSMP (timestamp) field of the standard header from the response control message shall contain a valid timestamp. The TMSP field is used for transmitting the times tamp in the response. Can be used for time calculation algorithm. Shall be answered as fast as possible by the Dlt module. If DltImplementTimestamp is not set NOT_SUPPORTED shall be the response.		

⌋(SRS_Dlt_00028)

7.7.7.1.19 Use ECU ID

[SWS_Dlt_00209] ⌈

Service name:	SetUseECUID		
Service ID [hex]	0x0D		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_status***	uint8	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	SetUsed to enable/disable the transmission of the ECU in the header.		

⌋()

[SWS_Dlt_00210] ⌈This message modifies DltHeaderUseEcuId. ⌋()

[SWS_Dlt_00506] ⌈

Service name:	GetUseECUID		
Service ID [hex]	0x1B		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***ecuid_status***	uint8	0 – OFF 1 – ON
Description:	Used to get status of DltHeaderUseEcuId		

⌋()

7.7.7.1.20 Use Session ID

[SWS_Dlt_00211] ⌈

Service name:	SetUseSessionID		
Service ID [hex]	0x0E		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_status***	uint8	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Used to enable/disable the transmission of the Session ID in the header.		

⌋()

[SWS_Dlt_00212] ⌈This message modifies DltHeaderUseSessionID. ⌋()

[SWS_Dlt_00507] ⌈

Service name:	GetUseSessionID		
Service ID [hex]	0x1C		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***sid_status***	uint8	0 – OFF 1 – ON
Description:	Used to get the status of DltHeaderUseSessionID.		

⌋()

7.7.7.1.21 Use Timestamp

[SWS_Dlt_00213] ⌈

Service name:	UseTimestamp
Service ID [hex]	0x0F
:	
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant

Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_status***	uint8	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Used to enable/disable the transmission of the Timestamp in the header.		

⌋()

[SWS_Dlt_00214] ⌈This message modifies DltHeaderUseTimestamp. ⌋()

[SWS_Dlt_00508] ⌈

Service name:	GetUseTimestamp		
Service ID [hex]	0x1D		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
none			
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***tst_status***	uint8	0 – OFF 1 – ON
Description:	Used to get the DltHeaderUseTimestamp.		

⌋()

7.7.7.1.22 Use Extended Header

[SWS_Dlt_00215] ⌈

Service name:	UseExtendedHeader		
Service ID [hex]	0x10		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***new_status***	uint8	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**

1	***Status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Used to enable/disable the transmission of the extended header. If DltImplementExtendedHeader is not set NOT_SUPPORTED shall be the response.		

⌋()

[SWS_Dlt_00216]⌈This message modifies DltHeaderUseExtendedHeader.⌋()

[SWS_Dlt_00509]⌈

Service name:	GetUseExtendedHeader		
Service ID [hex]	`0x1E`		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
none			
Response Parameter			
Number	***Name***	**`Type`**	**Description**
1	***Status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***exh_status***	`uint8`	0 – OFF 1 – ON
Description:	Used get the status of DltHeaderUseExtendedHeader		

⌋()

7.7.7.1.23 Call SW-C Injection

[SWS_Dlt_00217]⌈If the configuration parameter DltImplementSWCInjection is enabled Dlt module shall forward CallSW-CInjection messages to the according SW-C.
The Service_ID 0xFFF to 0xFFFFFFFF are reserved for this purpose. The value is user defined and can be freely used by a SW-C.⌋()

[SWS_Dlt_00218]⌈In the case of a CallSW-CInjection message, the Application ID (APID), Context ID (CTID) and the Session ID (SEID) shall be filled in the header. The pair of APID and CTID together with the SEID (see Session ID in 7.6.4) identifies a unique client server interface of a SW-C/runnable which is called in respect to reception of this message with the provided data (see Dlt_InjectCall).⌋()

[SWS_Dlt_00219]⌈If a unique identification is not possible (this pair does not exist, is not registered yet) the response shall be NOT_SUPPORTED.⌋()

[SWS_Dlt_00220] ⌈

Service name:	CallSW-CInjection		
Service ID [hex]	0xFFF ... 0xFFFFFFFF		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***data_length***	uint32	length of the provided data
2	***data[]***	uint8[]	data to provide to the SW-C
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR 3 == PENDING
Description:	Used to call a function in a SW-C. If DltImplementSWCInjection is not set NOT_SUPPORTED shall be the response.		

⌋ ()

7.7.7.1.24 Get ECU Software Version

[SWS_Dlt_00393] ⌈

Service name:	GetSoftwareVersion		
Service ID [hex]	0x13		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***length***	uint32	length of the string sw_version
3	***sw_version***	char[]	String containing the ECU – Software Version
Description:	This messages is for getting the SW-Version Number of the ECUs software. This shall be a string containing some numbers, dots, slashes or lines for identifying uniquely a software version number. The string is freely usable and mostly depends on the build system. In Non Verbose Mode his string shall be used to associate and identify a correct description file to the transmitted data. Because Message IDs and its associated arguments can vary on different SW versions a correct mapping of SW-version and description file is very important. In the associated description file this string for SW-version shall also be enclosed.		

⌋ ()

7.7.7.1.25 MessageBufferOverflow

[SWS_Dlt_00487] ⌈

Service name:	MessageBufferOverflow		
Service ID [hex]	0x14		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
Response Parameter			
Number	***Name***	**Type**	**Description**
1	***Status***	uint8	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***status***	uint8	0 – no Message BufferOverflow 1 – MessageBufferOverflow was happened
Description:	Used to tell that a MessageBufferOverflow happens. This can be requested by a client. Additionally this message is actively send by Dlt at the point when a MessageBufferOVervlow happens.		

⌋ ()

7.7.7.2 Timing messages

[SWS_Dlt_00221] ⌈The Dlt module shall send periodically time messages. This time messages are used to calculate a relative time on the external client and as alive signal. The period in which the timing messages are sent shall be one second. Timing messages are send by the Dlt module without request of an external client, but only if a connection to an external client is established (e.g. a diagnostic session with ROE or an interface of the Dlt communication module is enabled).⌋ (SRS_Dlt_00028)

[SWS_Dlt_00222] ⌈A timing message contains no data but the standard and the extended header. The SEID, APID and CTID fields can be left blank. All other fields shall be filled with its standard values, especially the Timestamp field. ⌋ ()

7.7.8 Connection management

To know the SW-Version of the ECU and to associate the correct description file to the received data an external client shall request the SW-Version from the Dlt module. Additionally the Dlt module than knows that a new external client has connected and can empty the message buffer.

[SWS_Dlt_00394] ⌈Each time a connection is established to a Dlt module the control message “Get ECU Software Version” shall be send from the external client to the Dlt module.

A connection is established if an external client is ready to receive data. This can be the opening of a TCP/IP connection or the starting on listening on a serial interface on the external client. ⌋()

[SWS_Dlt_00395] ⌈If the Dlt module receives a “Get ECU Software Version” control message, it shall first of all send the response to the "Get ECU SW Version" request. This should be done for storing the information about the running software of the ECU in the file on the external client side as first information (first message in file). ⌋()

[SWS_Dlt_00492] ⌈After the sending of the response to “ECU Software Version” the Dlt module shall empty its messages buffers by sending the containing messages over the network. ⌋()

[SWS_Dlt_00493] ⌈If the message buffer size (DltMessageBufferSize) is zero (0 byte) no message buffer shall be used. Instead, every time the messages shall be directly written to the interface, which is configured.

The disabling of the usage of message buffering can be done when a connection less interface like a serial interface is used and the external client is permanently listening. ⌋()

7.8 Error classification

[SWS_Dlt_00447] ⌈

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service called with wrong parameter	Development	DLT_E_WRONG_PARAMETERS	0x01
Provided API services of other modules returned with an error.	Development	DLT_E_ERROR_IN_PROV_SERVICE	0x02
The Dlt communication module detects an error in communication with its external interfaces	Development	DLT_E_COM_FAILURE	0x03
To many contexts are registered with the Dlt module. (e.g. the configuration tables are full)	Development	DLT_E_ERROR_TO_MANY_CONTEXT	0x04
The internal message	Development	DLT_E_MSG_LOOSE	0x05

buffer is full and the oldest messages are overwritten			
API service called with a NULL pointer. In case of this error, the API service shall return immediately without any further action, beside reporting this development error.	Development	DLT_E_PARAM_POINTER	0x06

⌋(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327)

7.9 Scheduling strategy

[SWS_Dlt_00468] ⌈The Dlt Module works completely event driven. The callback routines (the provide API for other modules and SW-C) generating this events. ⌋(SRS_BSW_00172)

8 API specification

8.1 Overview

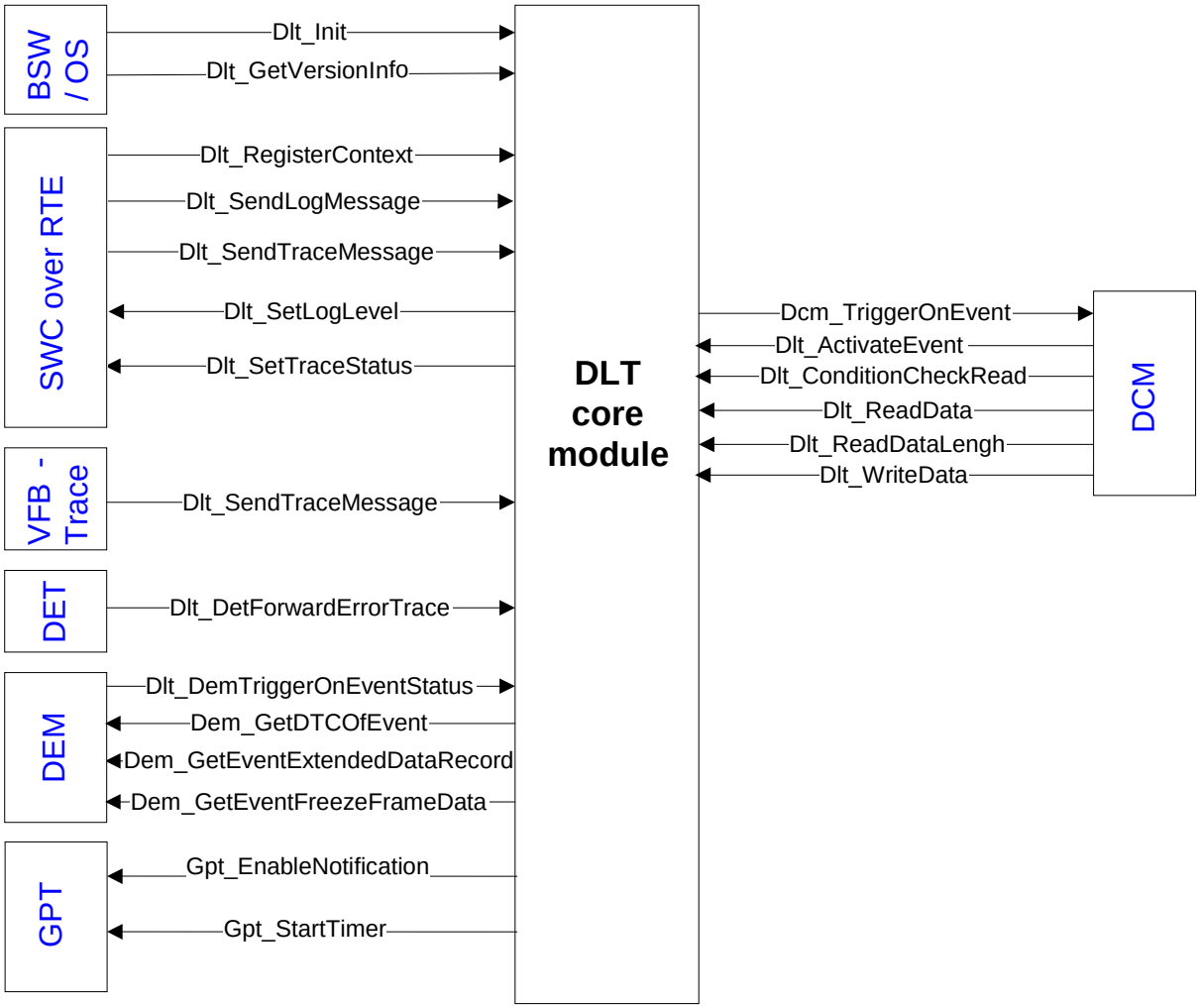


Figure 20 Overview of Dlt communication interfaces

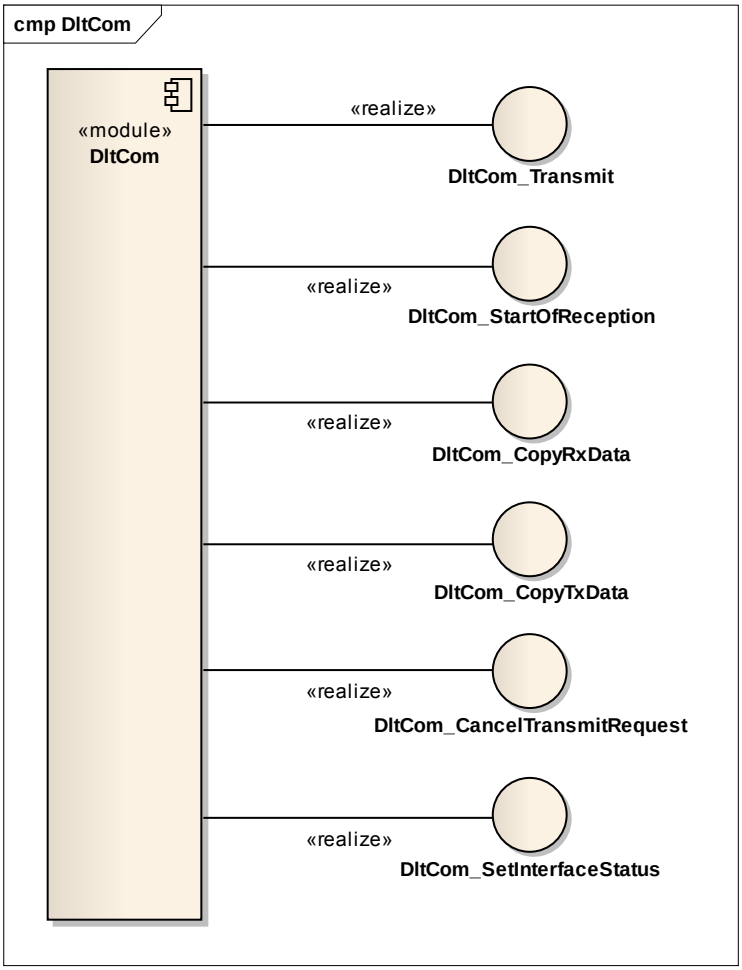


Figure 21 Internal interfaces between Dlt core module and Dlt communication module. The implementation of the Dlt communication module is not scope of this specification. The implementation of Dlt communication module is can be OEM or ECU specific.

8.1.1 Imported types

In this chapter all types included from the following files are listed:

Module	*Imported Type*
ComStack_Types	PduIdType
	PduInfoType
Dcm	Dcm_NegativeResponseCodeType
	Dcm_OpStatusType
	Dcm_RoeStateType
Dem	Dem_EventIdType
	Dem_UdsStatusByteType
Gpt	Gpt_ChannelType
	Gpt_ValueType
Std_Types	Std_ReturnType
	Std_VersionInfoType

In this chapter all types included from the following files are listed. The standard AUTOSAR types are defined in the AUTOSAR Specification of Standard Types document [13].

8.2 Service Interfaces

In this chapter all service interfaces are listed.

8.2.1 Client-Server-Interfaces

8.2.1.1 InjectionCallback

[SWS_Dlt_00498] ⌈

Name	InjectionCallback	
Comment	--	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

InjectCall		
Comments	--	
Variation	--	
Parameters	app_id	
	Comment	--
	Type	Dlt_ApplicationIDType
	Variation	--
	Direction	IN
	context_id	
	Comment	--
	Type	Dlt_ContextIDType
	Variation	--
	Direction	IN
	service_id	
	Comment	--

	Type	uint32
	Variation	--
	Direction	IN
	data_length	
	Comment	--
	Type	uint32
	Variation	--
	Direction	IN
	data	
	Comment	--
	Type	uint8*
	Variation	--
	Direction	IN
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--

⌋()

8.2.1.2 LogTraceSessionControl

[SWS_Dlt_00496] ⌈

Name	LogTraceSessionControl	
Comment	--	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

SetLogLevel	
Comments	--
Variation	--
Parameters	app_id

	Comment	--
	Type	Dlt_ApplicationIDType
	Variation	--
	Direction	IN
	context_id	
	Comment	--
	Type	Dlt_ContextIDType
	Variation	--
	Direction	IN
	loglevel	
	Comment	--
	Type	Dlt_MessageLogLevelType
	Variation	--
	Direction	IN
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--
SetTraceStatus		
Comments	--	
Variation	--	
Parameters	app_id	
	Comment	--
	Type	Dlt_ApplicationIDType
	Variation	--
	Direction	IN
	context_id	
	Comment	--
	Type	Dlt_ContextIDType
	Variation	--
	Direction	IN

	new_trace_status	
	Comment	--
	Type	boolean
	Variation	--
	Direction	IN
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--

⌋()

8.2.1.3 DLTService

[SWS_Dlt_00495] ⌈

Name	DLTService	
Comment	--	
IsService	true	
Variation	--	
Possible Errors	0	DLT_E_OK
	1	DLT_E_MSG_TOO_LARGE
	2	DLT_E_CONTEXT_ALREADY_REG
	3	DLT_E_UNKNOWN_SESSION_ID
	4	DLT_E_IF_NOT_AVAILABLE
	8	DLT_E_NOT_IN_VERBOSE_MODE

Operations

RegisterContext		
Comments	--	
Variation	--	
Parameters	session_id	
	Comment	--
	Type	Dlt_SessionIDType
	Variation	--
	Direction	IN
	app_id	

	Comment	--
	Type	Dlt_ApplicationIDType
	Variation	--
	Direction	IN
	context_id	
	Comment	--
	Type	Dlt_ContextIDType
	Variation	--
	Direction	IN
	description	
	Comment	--
	Type	uint8*
	Variation	--
	Direction	IN
	len_description	
	Comment	--
	Type	uint8
	Variation	--
	Direction	IN
Possible Errors	DLT_E_OK	Operation successful
	DLT_E_CONTEXT_ALREADY_REG	The software module context has already registered.
SendLogMessage		
Comments	--	
Variation	--	
Parameters	session_id	
	Comment	--
	Type	Dlt_SessionIDType
	Variation	--

	Direction	IN
	log_info	
	Comment	--
	Type	Dlt_MessageLogInfoType
	Variation	--
	Direction	IN
	log_data	
	Comment	--
	Type	uint8*
	Variation	--
	Direction	IN
	log_data_length	
	Comment	--
	Type	uint16
	Variation	--
	Direction	IN
Possible Errors	DLT_E_OK	Operation successful
	DLT_E_MSG_TOO_LARGE	The message is too big for the internal Dlt buffer.
	DLT_E_UNKNOWN_SESSION_ID	The provided session id is unknown.
	DLT_E_IF_NOT_AVAILABLE	The interface is not available.
	DLT_E_NOT_IN_VERBOSE_MODE	Unable to send the message in verbose mode.
SendTraceMessage		
Comments	--	
Variation	--	
Parameters	session_id	
	Comment	--
	Type	Dlt_SessionIDType
	Variation	--

	Direction	IN
	trace_info	
	Comment	--
	Type	Dlt_MessageTraceInfoType
	Variation	--
	Direction	IN
	trac_data	
	Comment	--
	Type	uint8*
	Variation	--
	Direction	IN
	trace_data_length	
	Comment	--
	Type	uint16
	Variation	--
	Direction	IN
Possible Errors	DLT_E_OK	Operation successful
	DLT_E_MSG_TOO_LARGE	The message is too big for the internal Dlt buffer.
	DLT_E_UNKNOWN_SESSION_ID	The provided session id is unknown.
	DLT_E_IF_NOT_AVAILABLE	The interface is not available.
	DLT_E_NOT_IN_VERBOSE_MODE	Unable to send the message in verbose mode.

⌋()

8.2.1.4 VerboseModeControl

[SWS_Dlt_00497] ⌈
Only connected by Dlt if DltImplementVerboseMode is set.

Name	VerboseModeControl
Comment	--
IsService	true

Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

SetVerboseMode		
Comments	--	
Variation	--	
Parameters	app_id	
	Comment	--
	Type	Dlt_ApplicationIDType
	Variation	--
	Direction	IN
	context_id	
	Comment	--
	Type	Dlt_ContextIDType
	Variation	--
	Direction	IN
	is_verbose_mode	
	Comment	--
	Type	boolean
	Variation	--
	Direction	IN
Possible Errors	E_OK	Operation successful
	E_NOT_OK	--

⌋()

8.2.2 Ports

8.2.2.1 Dlt_ICN

Name	ICN		
Kind	RequiredPort	Interface	InjectionCallback

Description	--
Variation	--

8.2.2.2 Dlt_PSCN

Name	PSCN		
Kind	RequiredPort	Interface	LogTraceSessionControl
Description	--		
Variation	--		

8.2.2.3 Dlt_service

[SWS_Dlt_00499] ⌈

Name	service		
Kind	ProvidedPort	Interface	DLTService
Description	--		
Variation	--		

⌋ ()

8.2.2.4 Dlt_VCN

Name	VCN		
Kind	RequiredPort	Interface	VerboseModeControl
Description	--		
Variation	--		

8.3 Type definitions

8.3.1 Dlt_ConfigType

[SWS_Dlt_00437] ⌈

Name:	`Dlt_ConfigType`	
Type:	`Structure`	
Range:	`implementation specific`	The content of the initialization data structure is implementation specific
Description:	This is the type of the data structure containing the initialization data for Dlt.	

⌋ (SRS_BSW_00414)

8.3.2 Dlt_MessageTypeType

[SWS_Dlt_00224] ⌈

Name:	`Dlt_MessageTypeType`	
Type:	`Enumeration`	
Range:	`DLT_TYPE_LOG`	0x1: A log message
	`DLT_TYPE_APP_TRACE`	0x2: A trace message
	`DLT_TYPE_NW_TRACE`	0x3: A message forwarded from a communication bus (like CAN, FlexRay)
	`DLT_TYPE_CONTROL`	0x4: A message for internal use/control send between Dlt module and external client.
Description:	This type describes the type of the message.	

⌋ ()

8.3.3 Dlt_SessionIDType

[SWS_Dlt_00225] ⌈

Name:	`Dlt_SessionIDType`		
Type:	`uint8, uint16, uint32`		
Range:	`<range of legal values>`	--	is platform depended, can be set individually (possible values are uint8, uint16 and uint32)
Description:	This type describes the Session ID		

⌋ ()

8.3.4 Dlt_ApplicationIDType

[SWS_Dlt_00226] ⌈

Name:	`Dlt_ApplicationIDType`		
Type:	`uint8, uint16, uint32`		
Range:	`<range of legal values>`	--	is platform depended, can be set individual
Description:	This type describes the Application ID		

⌋ ()

8.3.5 Dlt_ContextIDType

[SWS_Dlt_00227] ⌈

Name:	`Dlt_ContextIDType`		
Type:	`uint8, uint16, uint32`		
Range:	`<range of legal values>`	--	is platform depended, can be set individual
Description:	This type describe the Context ID		

⌋ ()

8.3.6 Dlt_MessageIDType

[SWS_Dlt_00228] ⌈

Name:	Dlt_MessageIDType		
Type:	uint8, uint16, uint32		
Range:	<range of legal values>	--	is platform depended, can be set individual
Description:	contains the unique Message ID for a message		

⌋ ()

8.3.7 Dlt_MessageOptionsType

[SWS_Dlt_00229] ⌈

Name:	Dlt_MessageOptionsType		
Type:	uint8		
Range:	verbose_mode	--	Bit 3: If set Verbose mode is used (yet not relevant)
	message_type	--	Bit 0-2 Dlt_MessageTypeType: determines type of msg (log,trace,...)
Description:	Bitfield		

⌋ ()

8.3.8 Dlt_MessageLogLevelType

[SWS_Dlt_00230] ⌈

Name:	Dlt_MessageLogLevelType	
Type:	Enumeration	
Range:	DLT_LOG_OFF	0x00: Turn off logging
	DLT_LOG_FATAL	0x01: Fatal system error
	DLT_LOG_ERROR	0x02: Errors occurring in a SW-C with impact to correct functionality
	DLT_LOG_WARN	0x03 : Log messages where a incorrect behavior can not be ensured
	DLT_LOG_INFO	0x04 : Log messages providing information for better understanding of the internal behavior of a software
	DLT_LOG_DEBUG	0x05 : Log messages, which are usable only for debugging of a software
	DLT_LOG_VERBOSE	0x06 : Log messages with the highest communicative level, here all possible states, information and everything else can be logged
Description:	This type describes the log level for each log message.	

⌋ ()

8.3.9 Dlt_MessageTraceType

[SWS_Dlt_00231] ⌈

Name:	Dlt_MessageTraceType	
Type:	Enumeration	
Range:	DLT_TRACE_VARIABLE	0x01: For tracing the value of a variable

	DLT_TRACE_FUNCTION_IN	0x02: For tracing the calling of a function
	DLT_TRACE_FUNCTION_OUT	0x03: For tracing the returning of a function
	DLT_TRACE_STATE	0x04: For tracing a state of a state machine
	DLT_TRACE_VFB	0x05: For tracing RTE Events
Description:	This type describes labels for trace messages.	

⌋ ()

8.3.10 Dlt_MessageControlInfoType

[SWS_Dlt_00232] ⌈

Name:	Dlt_MessageControlInfoType	
Type:	Enumeration	
Range:	DLT_CONTROL_REQUEST	0x01 : Declares a request of an operation on Dlt module
	DLT_CONTROL_RESPONSE	0x02 : Declares a the answer of an request
	DLT_CONTROL_TIME	0x03 : Declares a timing message
Description:	This type describes the type of a Dlt control message.	

⌋ ()

8.3.11 Dlt_MessageNetworkTraceInfoType

[SWS_Dlt_00233] ⌈

Name:	Dlt_MessageNetworkTraceInfoType	
Type:	Enumeration	
Range:	DLT_NW_TRACE_IPC	0x01 : Inter process communication
	DLT_NW_TRACE_CAN	0x02 : CAN communication
	DLT_NW_TRACE_FLEXRAY	0x03 : Flexray communication
Description:	This type describes transported type of a Dlt BUSMESSAGE.	

⌋ ()

8.3.12 Dlt_ MessageArgumentCount

[SWS_Dlt_00235] ⌈

Name:	Dlt_MessageArgumentCount
Type:	uint16
Description:	This type describe count of arguments provided to a message (only used if DltImplementVerboseMode is set)

⌋ ()

8.3.13 Dlt_MessageLogInfoType

[SWS_Dlt_00236] ⌈

Name:	Dlt_MessageLogInfoType		
Type:	Structure		
Element:	Dlt_MessageArgumentCount	arg_count	only used if DltImplementVerboseMode is set

	Dlt_MessageLogLevelType	log_level	--
	Dlt_MessageOptionsType	options	--
	Dlt_ContextIDType	context_id	--
	Dlt_ApplicationIDType	app_id	--
Description:	--		

⌋ ()

8.3.14 Dlt_MessageTraceInfoType

[SWS_Dlt_00237] ⌈

Name:	Dlt_MessageTraceInfoType		
Type:	Structure		
Element:	Dlt_MessageTraceType	trace_info	--
	Dlt_MessageOptionsType	options	--
	Dlt_ContextIDType	context	--
	Dlt_ApplicationIDType	app_id	--
Description:	--		

⌋ ()

8.3.15 Dlt_ReturnType

[SWS_Dlt_00238] ⌈

Name:	Dlt_ReturnType		
Type:	Std_ReturnType		
Range:	DLT_E_OK	0	The required operation succeeded.
	DLT_E_MSG_TOO_LARGE	1	The message is too big for the internal Dlt buffer.
	DLT_E_CONTEXT_ALREADY_REG	2	The software module context has already registered.
	DLT_E_UNKNOWN_SESSION_ID	3	Unknown session id.
	DLT_E_IF_NOT_AVAILABLE	4	The interface is not available.
	DLT_E_IF_BUSY	5	The interface is busy.
	DLT_E_ERROR_UNKNOWN	6	An unknown error is occurred.
	DLT_E_PENDING	7	Application process not yet completed, another call is required.
	DLT_E_NOT_IN_VERBOSE_MOD	8	The Verbose Mode DltImplementVerboseMode flag has not been set, hence all functions that requires to send a message in verbose mode (i.e. Dlt_SendLogMessage and Dlt_SendTraceMessage) shell return this error.
Description:	--		

⌋ (SRS_BSW_00377)

8.4 Function definitions

This is a list of functions provided for upper layer modules.

8.4.1 General provided Functions for BSW-modules

8.4.1.1 Dlt_Init

[SWS_Dlt_00239] ⌈

Service name:	Dlt_Init

Syntax:	`void Dlt_Init(`
	`  const Dlt_ConfigType* config`
	`)`

Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	config: Pointer to a DLT configuration structure
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Dlt is using the NVRamManager and is to be initialized very late in the ECU startup phase. The Dlt_Init() function should be called after the NVRamManager is initialed."

⌋ (SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00101, SRS_BSW_00407, SRS_BSW_00358, SRS_BSW_00414)

8.4.1.2 Dlt_GetVersionInfo

[SWS_Dlt_00271] ⌈

Service name:	Dlt_GetVersionInfo

Syntax:	`void Dlt_GetVersionInfo(`
	`  Std_VersionInfoType* versioninfo`
	`)`

Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	versioninfo: Pointer to where to store the version information of this module.
Return value:	None
Description:	Returns the version information of this module.

⌋ (SRS_BSW_00402)

8.4.2 Provided functions for sending log messages from SW-Cs

8.4.2.1 Dlt_SendLogMessage

[SWS_Dlt_00241] ⌈

Service name:	Dlt_SendLogMessage

Syntax:	`Dlt_ReturnType Dlt_SendLogMessage(`
	`  Dlt_SessionIDType session_id,`
	`  const Dlt_MessageLogInfoType* log_info,`
	`  const uint8* log_data,`
	`  uint16 log_data_length`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	session_id	For SW-C this is not visible (Port defined argument value), for BSW-modules it is the module number
	log_info	Structure containing the relevant information for filtering the message.
	log_data	Buffer containing the parameters to be logged. The contents of this pointer represents the payload of the send log message
	log_data_length	Length of the data buffer log_data.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Dlt_ReturnType	DLT_E_OK - The required operation succeeded. DLT_E_MSG_TOO_LARGE - The message is too large for sending over the network. DLT_E_IF_NOT_AVAILABLE - The interface is not available. DLT_E_UNKNOWN_SESSION_ID - The provided session id is unknown. DLT_E_NOT_IN_VERBOSE_MODE - Unable to send the message in verbose mode.
Description:	The service represents the interface to be used by basic software modules or by software component to send log messages.	

⌋(SRS_Dlt_00003)

[SWS_Dlt_00242] ⌈The payload (log_data) shall contain the complete payload of the Dlt log message (compare chapter 7.7.5). This means that the structure and the contents of the provided memory in log_data shall completely compliant to the Dlt protocol payload specification. ⌋()

8.4.2.2 Dlt_SendTraceMessage

[SWS_Dlt_00243] ⌈

Service name:	Dlt_SendTraceMessage	

Syntax:	`Dlt_ReturnType Dlt_SendTraceMessage(`
	`  Dlt_SessionIDType session_id,`
	`  const Dlt_MessageTraceInfoType* trace_info,`
	`  const uint8* trace_data,`
	`  uint16 trace_data_length`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	session_id	--
	trace_info	Structure containing the relevant information for filtering the message.

	trace_data	Buffer containing the parameters to be traced. The contents of this pointer represents the payload of the send log message
	trace_data_length	Length of the data buffer trace_data
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Dlt_ReturnType	DLT_E_OK - The required operation succeeded. DLT_E_MSG_TOO_LARGE - The message is too large for sending over the network. DLT_E_IF_NOT_AVAILABLE - The interface is not available. DLT_E_UNKNOWN_SESSION_ID - The provided session id is unknown. DLT_E_NOT_IN_VERBOSE_MODE - Unable to send the message in verbose mode.
Description:	The service represents the interface to be used by basic software modules or by software component to trace parameters.	

⌋(SRS_Dlt_00003)

[SWS_Dlt_00244] ⌈The payload (trace_data) shall contain the complete payload of the Dlt log message (compare chapter 7.7.5). This means that the structure and the contents of the provided memory in trace_data shall completely compliant to the Dlt protocol payload specification.⌋()

8.4.2.3 Dlt_RegisterContext

[SWS_Dlt_00245] ⌈

Service name:	Dlt_RegisterContext	

Syntax:	`Dlt_ReturnType Dlt_RegisterContext(`
	`  Dlt_SessionIDType session_id,`
	`  Dlt_ApplicationIDType app_id,`
	`  Dlt_ContextIDType context_id,`
	`  const uint8* app_description,`
	`  uint8 len_app_description,`
	`  const uint8* context_description,`
	`  uint8 len_context_description`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	session_id	number of the module (Module ID within BSW, Port defined argument value within SW-C)
	app_id	the Application ID
	context_id	the Context ID
	app_description	Points to description string for the provided application id. At maximum 255 characters are interpreted.
	len_app_description	The length of the description for the application id string (number of characters of description string).
	context_description	Points to description string for the provided context. At maximum 255 characters are interpreted.
	len_context_description	The length of the description string (number of characters of description string).
Parameters	None	

(inout):		
Parameters (out):	None	
Return value:	Dlt_ReturnType	DLT_E_CONTEXT_ALREADY_REG - The software module context has already registered. DLT_E_OK - The required operation succeed.
Description:	The service has to be called when a software module wants to use services offered by DLT software component for a specific context. If a context id is being registered for an already registered application id then app_description can be NULL and len_app_description zero.	

⌋ (SRS_Dlt_00033)

8.4.2.4 Provided functions for triggering DTC changes from Dem

8.4.2.5 Dlt_DemTriggerOnEventStatus

[SWS_Dlt_00470] ⌈

Service name:	Dlt_DemTriggerOnEventStatus	

Syntax:	`void Dlt_DemTriggerOnEventStatus(`
	`  Dem_EventIdType EventId,`
	`  Dem_UdsStatusByteType EventStatusByteOld,`
	`  Dem_UdsStatusByteType EventStatusByteNew`
	`)`	

Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	EventId	Identification of an Event by assigned event number. The Event Number is configured in the Dem. Min.: 1 (0: Indication of no Event or Failure) Max.: Result of configuration of Event Numbers in Dem (Max is either 255 or 65535)
	EventStatusByteOld	Extended event status before change
	EventStatusByteNew	Detected / reported of event status
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service is provided by the Dem in order to call Dlt upon status changes.	

⌋ (SRS_Dlt_00007)

8.4.3 Provided function for fan-out capability of Det

8.4.3.1 Dlt_DetForwardErrorTrace

[SWS_Dlt_00432] ⌈

Service name:	Dlt_DetForwardErrorTrace
Syntax:	`void Dlt_DetForwardErrorTrace(` `  uint16 ModuleId,`

	uint8 InstanceId, uint8 ApiId, uint8 ErrorId)	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ModuleId	Module ID of calling module.
	InstanceId	The identifier of the index based instance of a module, starting from 0, If the module is a single instance module it shall pass 0 as the InstanceId.
	ApiId	ID of API service in which error is detected (defined in SWS of calling module)
	ErrorId	ID of detected development error (defined in SWS of calling module).
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service to forward error reports from Det to Dlt.	

⌋ (SRS_Dlt_00006)

8.4.4 Provided interfaces for Dcm

8.4.4.1 Dlt_ActivateEvent

[SWS_Dlt_00488] ⌈

Service name:	Dlt_ActivateEvent	
Syntax:	Std_ReturnType Dlt_ActivateEvent(uint8 RoeEventID, Dcm_RoeStateType RoeState)	
Service ID[hex]:	0x11	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	RoeEventID	The eventID to use for the ResponseOnEvent (0x86). This eventId shall be used within the Dcm_TriggerOnEvent() function called by Dlt."
	RoeState	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK - call succeeded E_NOT_OK - call has some errors
Description:	This function is called by the Dcm if a specific ResponseOnEvent is enabled by a user. The RoeEventID shall be used by the Dlt to notify the Dcm about some actions to do for the ROE service. Normally for the Dlt module, only the ReadDataByIdendifer (0x22) should be used for ROE. In addition, when a specific ROE for the Dlt module is disabled/turned off by a user, the Dlt module is notified with this function.	

⌋ ()

8.4.4.2 Dlt_ReadData

[SWS_Dlt_00247] ⌈

Service name:	Dlt_ReadData	

Syntax:	`Std_ReturnType Dlt_ReadData(`
	`  Dcm_OpStatusType OpStatus,`
	`  uint8* data`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	OpStatus	See description of Dcm_OpStatusType in AUTOSAR_SWS_DCM.
Parameters (inout):	None	
Parameters (out):	data	Contains a complete Dlt message
Return value:	Std_ReturnType	E_OK - call succeeded E_PENDING - application process not yet completed, another call is required E_NOT_OK - call has some errors
Description:	Is called from Dcm when UDS service ReadDataByIdendifer for Dlt DID is called.	

⌋ ()

8.4.4.3 Dlt_ReadDataLength

[SWS_Dlt_00248] ⌈

Service name:	Dlt_ReadDataLength	

Syntax:	`Std_ReturnType Dlt_ReadDataLength(`
	`  uint16* DataLength`
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	DataLength	Data length of the DID to read by Dcm (This is the Dlt message length since only Dlt message are transmitted)
Return value:	Std_ReturnType	E_OK - call succeeded E_NOT_OK - call has some errors
Description:	Is called from Dcm when UDS service ReadDataByIdendifer for Dlt DID is called.	

⌋ ()

8.4.4.4 Dlt_WriteData

[SWS_Dlt_00249] ⌈

Service name:	Dlt_WriteData	

Syntax:	`Std_ReturnType Dlt_WriteData(`
	`  uint8* data,`
	`  uint16 dataLength,`
	`  Dcm_OpStatusType OpStatus,`
	`  Dcm_NegativeResponseCodeType* ErrorCode`
	`)`	

Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	data	data to write (Shall contain a complete Dlt message send from a external client by using WriteDataByIdendifer)
	dataLength	length of data to write by Dcm
	OpStatus	See description of Dcm_OpStatusType in AUTOSAR_SWS_DCM.
Parameters (inout):	None	
Parameters (out):	ErrorCode	--
Return value:	Std_ReturnType	E_OK - call succeeded E_PENDING - application process not yet completed, another call is required E_NOT_OK - call has some errors
Description:	Is called from Dcm when UDS service WriteDataByIdendifer for Dlt DID is called.	

⌋ ()

8.4.4.5 Dlt_ConditionCheckRead

[SWS_Dlt_00428] ⌈

Service name:	Dlt_ConditionCheckRead	

Syntax:	`Std_ReturnType Dlt_ConditionCheckRead(`
	`  Dcm_OpStatusType OpStatus,`
	`  Dcm_NegativeResponseCodeType* ErrorCode`
	`)`	

Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	OpStatus	See description of Dcm_OpStatusType in AUTOSAR_SWS_DCM.
Parameters (inout):	None	
Parameters (out):	ErrorCode	See description in Dcm service component
Return value:	Std_ReturnType	E_OK - call succeeded E_PENDING - application process not yet completed, another call is required E_NOT_OK - call has some errors
Description:	Is called from Dcm when UDS service ReadDataByIdendifer for Dlt DID is called to see if Dlt_ReadData can be called.	

⌋ ()

8.4.5 Interfaces provided by Dlt core module for internal use with Dlt communication module

8.4.5.1 Dlt_ComRxIndication

[SWS_Dlt_00272] ⌈

Service name:	Dlt_ComRxIndication	

Syntax:	`void Dlt_ComRxIndication(`
	`  PduIdType DltRxPduId,`
	`  Std_ReturnType Result`
	`)`	

Service ID[hex]:	0x0f	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	DltRxPduId	ID of DLT I-PDU that has been received. Identifies the data that has been received. Range: 0..(maximum number of I-PDU IDs received by DLT) 1
	Result	Result of the N-PDU reception: - E_OK: the complete N-PDU has been received. - E_NOT_OK: an error occurred during reception, used to enable unlocking of the receive buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This function is called by the Dlt communication module: - with Result = E_OK after the complete Dlt I-PDU has successfully been received, i.e. at the very end of the segmented TP receive cycle or after receiving an unsegmented N-PDU. - with Result = E_NOT_OK it is indicated that an error (e.g. timeout) has occurred during the reception of the Dlt I-PDU. This passes the receive buffer back to Dlt and allows error handling. It is undefined which part of the buffer contains valid data in this case, so Dlt shall not evaluate that buffer. By calling this service only Dlt is allowed to access the buffer. **Caveats:** This function might be called in interrupt context. If an existing reception has to be canceled to establish a new reception it is required to indicate a cancellation first before requesting a buffer for the new reception. Otherwise Dlt will be requested to open a second connection.	

⌋ (SRS_Dlt_00034)

8.4.5.2 Dlt_ComTxConfirmation

[SWS_Dlt_00273] ⌈

Service name:	Dlt_ComTxConfirmation

Syntax:	`void Dlt_ComTxConfirmation(`
	`  PduIdType DltTxPduId,`
	`  Std_ReturnType Result`
	`)`

Service ID[hex]:	0x10
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant	
Parameters (in):	DltTxPduId	ID of Dlt I-PDU that has been transmitted. Range: 0..(maximum number of I-PDU IDs transmitted by Dcm) - 1
	Result	Result of the N-PDU transmission: - E_OK: the complete N-PDU has been transmitted. - E_NOT_OK: an error occurred during transmission, used to enable unlocking of the transmit buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This function is called by the Dlt communication module: - with Result = E_OK after the complete Dlt I-PDU has successfully been transmitted, i.e. at the very end of the segmented TP transmit cycle. Within this function, Dlt shall unlock the transmit buffer. - with Result = E_NOT_OK if an error (e.g. timeout) has occurred during the transmission of the Dlt I-PDU. This enables unlocking of the transmit buffer. The I-PDU can be rejected and an error reporting may be done. Error handling is up to the PduRouter. - with Result = E_NOT_OK if the N-PDU has been successfully cancelled after a request with Dlt_ComCancelTransmitRequest() Caveats: This function might be called in interrupt context (e.g. from a transmit interrupt). However, for transmission via FlexRay there is a restriction: Since the FlexRay Specification does not mandate the existence of a transmit interrupt, the exact meaning of this confirmation (i.e. "transfer into the FlexRay controller's send buffer" OR "transmission onto the FlexRay network") depends on the capabilities of the FlexRay communication controller and the configuration of the FlexRay Interface.	

⌋(SRS_Dlt_00034)

8.5 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The names of these kind of interfaces is not fixed because they are configurable.

8.5.1 Expected Interfaces from SW-Cs

8.5.1.1 Dlt_SetLogLevel

[SWS_Dlt_00252] ⌈

Service name:	Dlt_SetLogLevel_<SESSION>

Syntax:	`void Dlt_SetLogLevel_<SESSION>(`
	`  Dlt_ApplicationIDType app_id,`
	`  Dlt_ContextIDType context_id,`
	`  Dlt_MessageLogLevelType loglevel`
	`)`

Service ID[hex]:	0x11
Sync/Async:	Asynchronous

Reentrancy:	Non Reentrant	
Parameters (in):	app_id	the Application ID
	context_id	the Context ID
	loglevel	the new log level of the context
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Callback is called by Dlt to inform the SW-C of a new log level status of a given context.	

⌋(SRS_Dlt_00004, SRS_Dlt_00038)

[SWS_Dlt_00253] ⌈ This function shall be provided by a SW-C and is called from Dlt when the log level for the given pair of Application ID and Context ID changes. ⌋()

8.5.1.2 Dlt_SetTraceStatus

[SWS_Dlt_00254] ⌈

Service name:	Dlt_SetTraceStatus_<SESSION>	

Syntax:	`void Dlt_SetTraceStatus_<SESSION>(`
	`  Dlt_ApplicationIDType app_id,`
	`  Dlt_ContextIDType context_id,`
	`  boolean new_trace_status`
	`)`	

Service ID[hex]:	0x12	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	app_id	the Application ID
	context_id	the Context ID
	new_trace_status	the new trace_status for the pair of Application ID and Context ID
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Callback is called by Dlt to inform the SW-C of a new trace status of a given context.	

⌋(SRS_Dlt_00004, SRS_Dlt_00038)

[SWS_Dlt_00255] ⌈This function shall be provided by a SW-C and is called from Dlt when the trace status for the given pair of Application ID and Context ID changes. ⌋()

8.5.1.3 Dlt_SetVerboseMode

[SWS_Dlt_00256] ⌈

Service name:	Dlt_SetVerboseMode_<SESSION>

Syntax:	`void Dlt_SetVerboseMode_<SESSION>(`
	`  Dlt_ApplicationIDType app_id,`
	`  Dlt_ContextIDType context_id,`
	`  boolean is_verbose_mode`
	`)`	

Service ID[hex]:	0x13	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	app_id	the Application ID
	context_id	the Context ID
	is_verbose_mode	if true, Verbose Mode is enabled, if false Verbose Mode is disabled
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Callback is called by Dlt to inform the SW-C of a change of the verbose mode.	

⌋()

[SWS_Dlt_00257] ⌈This interface shall only be called by Dlt if DltImplementVerboseMode is set. ⌋()

[SWS_Dlt_00258] ⌈This function shall be provided by a SW-C and is called from Dlt when the Verbose Mode changes. ⌋()

8.5.1.4 Dlt_InjectCall

[SWS_Dlt_00259] ⌈

Service name:	Dlt_InjectCall_<SESSION>	

Syntax:	`void Dlt_InjectCall_<SESSION>(`
	`  Dlt_ApplicationIDType app_id,`
	`  Dlt_ContextIDType context_id,`
	`  uint32 service_id,`
	`  uint32 data_length,`
	`  const uint8* data`
	`)`	

Service ID[hex]:	0x14	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	app_id	the Application ID
	context_id	the Context ID
	service_id	the service ID for the injection (user defined)
	data_length	length of the data puffer provided
	data	pointer to data puffer with data belonging to the injection (service ID). The contents of the data is user defined
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Callback is called by Dlt to inject a function call in the SW-C. The behaviour trigged by this function should depend on the service_id also the interpretation of the user data. Both are specific to the application.	

⌋()

[SWS_Dlt_00260] ⌈This interface shall only called by Dlt when DltImplementSWCInjection is set.⌋()

[SWS_Dlt_00261] ⌈This function shall be provided by a SW-C and is called from Dlt when the Verbose Mode changes.⌋()

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Expected Interfaces from Dcm

8.6.1.1 Dcm_TriggerOnEvent

[SWS_Dlt_00262] ⌈

Service name:	Dcm_TriggerOnEvent	

Syntax:	`Std_ReturnType Dcm_TriggerOnEvent(`
	`  uint8 RoeEventId`
	`)`	

Service ID[hex]:	0x2D	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	RoeEventId	Identifier of the event that is triggered
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: RoeEventId value is valid E_NOT_OK: RoeEventId value is not valid
Description:	The call to this function allows to trigger an event linked to a ResponseOnEvent request. On the function call, the DCM will execute the associated service if the corresponding Mode of the RoeEventId is 'ROE started'.	

⌋()

8.6.2 Expected Interfaces from Dlt communication module

8.6.2.1 DltCom_CopyRxData

[SWS_Dlt_00515] ⌈

Service name:	DltCom_CopyRxData
Syntax:	`BufReq_ReturnType DltCom_CopyRxData(`

	PduIdType id, const PduInfoType* info, PduLengthType* bufferSizePtr)	
Service ID[hex]:	0x16	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the received I-PDU.
	info	Pointer to the buffer (SduDataPtr) and its length (SduLength) containing the data to be copied by the upper layer module.
Parameters (inout):	None	
Parameters (out):	bufferSizePtr	Available receive buffer after data has been copied.
Return value:	BufReq_ReturnType	BUFREQ_OK: Data copied successfully BUFREQ_E_NOT_OK: Data was not copied because an error occurred
Description:	This function is called to provide the received data of an I-PDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the I-PDU data. The size of the remaining data is written to the position indicated by bufferSizePtr.	

⌋(SRS_Dlt_00034)

8.6.2.2 DltCom_CopyTxData

[SWS_Dlt_00516] ⌈

Service name:	DltCom_CopyTxData	
Syntax:	BufReq_ReturnType DltCom_CopyTxData(PduIdType id, const PduInfoType* info, RetryInfoType* retry, PduLengthType* availableDataPtr)	
Service ID[hex]:	0x43	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the transmitted I-PDU.
	info	Provides the destination buffer and the number of bytes to be copied. If not enough transmit data is available, no data is copied. The transport protocol module may retry. A copy size of 0 can be used to indicate state changes in the retry parameter or to query currently available data.
	retry	This parameter is used to acknowledge transmitted data or to retransmit data after transmission problems. If the retry parameter is a NULL_PTR, it indicates that the transmit data can be removed from the buffer immediately after it has been copied. Otherwise, the retry parameter must point to a valid RetryInfoType element. If TpDataState indicates TP_CONFPENDING, the previously copied data must remain in the TP buffer to be available for error recovery. TP_DATACONF indicates that all data that has been copied

		before this call is confirmed and can be removed from the TP buffer. Data copied by this API call is excluded and will be confirmed later. TP_DATARETRY indicates that this API call shall copy previously copied data in order to recover from an error. In this case TxTpDataCnt specifies the offset in bytes from the current data copy position.
Parameters (inout):	None	
Parameters (out):	availableDataPtr	Indicates the remaining number of bytes that are available in the upper layer module's Tx buffer. availableDataPtr can be used by TP modules that support dynamic payload lengths (e.g. FrIsoTp) to determine the size of the following CFs.
Return value:	BufReq_ReturnType	BUFREQ_OK: Data has been copied to the transmit buffer completely as requested. BUFREQ_E_BUSY: Request could not be fulfilled, because the required amount of Tx data is not available. The lower layer module may retry this call later on. No data has been copied. BUFREQ_E_NOT_OK: Data has not been copied. Request failed.
Description:	This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the I-PDU data unless retry->TpDataState is TP_DATARETRY. In this case the function restarts to copy the data beginning at the offset from the current position indicated by retry->TxTpDataCnt. The size of the remaining data is written to the position indicated by availableDataPtr.	

⌋(SRS_Dlt_00034)

8.6.2.3 DltCom_StartOfReception

[SWS_Dlt_00517] ⌈

Service name:	DltCom_StartOfReception	

Syntax:	`BufReq_ReturnType DltCom_StartOfReception(`
	`  PduIdType id,`
	`  const PduInfoType* info,`
	`  PduLengthType TpSduLength,`
	`  PduLengthType* bufferSizePtr`
	`)`	

Service ID[hex]:	0x18	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the I-PDU.
	info	Pointer to a PduInfoType structure containing the payload data (without protocol information) and payload length of the first frame or single frame of a transport protocol I-PDU reception. Depending on the global parameter MetaDataLength, additional bytes containing MetaData (e.g. CAN ID) are appended after the payload data.
	TpSduLength	Total length of the N-SDU to be received.
Parameters (inout):	None	
Parameters (out):	bufferSizePtr	Available receive buffer in the receiving module. This

		parameter will be used to compute the Block Size (BS) in the transport protocol module.
Return value:	BufReq_ReturnType	BUFREQ_OK: Connection has been accepted. bufferSizePtr indicates the available receive buffer; reception is continued. If no buffer of the requested size is available, a receive buffer size of 0 shall be indicated by bufferSizePtr. BUFREQ_E_NOT_OK: Connection has been rejected; reception is aborted. bufferSizePtr remains unchanged. BUFREQ_E_OVFL: No buffer of the required length can be provided; reception is aborted. bufferSizePtr remains unchanged.
Description:	This function is called at the start of receiving an N-SDU. The N-SDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF).	

⌋ (SRS_Dlt_00034)

8.6.2.4 DltCom_Transmit

[SWS_Dlt_00263] ⌈

Service name:	DltCom_Transmit	

Syntax:	`Std_ReturnType DltCom_Transmit(`
	`  PduIdType DltTxPduId,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	DltTxPduId	ID of Dlt I-PDU to be transmitted. Range: 0..(maximum number of I-PDU IDs which may be transmitted by Dcm) - 1
	PduInfoPtr	Pointer to a structure with I-PDU related data that shall be transmitted: data length and pointer to I-SDU buffer
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Transmit request has been accepted E_NOT_OK: Transmit request has not been accepted
Description:	The Dlt communication module requests a transmission for the Dlt core module.	

⌋ (SRS_Dlt_00034)

8.6.2.5 DltCom_CancelTransmitRequest

[SWS_Dlt_00264] ⌈

Service name:	DltCom_CancelTransmitRequest	

Syntax:	`Std_ReturnType DltCom_CancelTransmitRequest(`
	`  PduIdType PduId`
	`)`	

Service ID[hex]:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	PduId	This parameter contains the unique identifier of the I-PDU which transfer has to be cancelled.

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	**E_NOT_OK:** Cancellation request of the transfer of the specified I-PDU is rejected, e. g. cancellation is requested at the receiver in an 1:n connection or in an unsegmented transfer at the receiver or cancellation is not allowed for the corresponding channel. **E_OK:** Cancellation request of the transfer (sending or receiving) of the specified I-PDU is accepted.
Description:	This service primitive is used to cancel the transfer of pending I-PDUs. This function has to be called with the PDU-Id and the reason for cancellation.	

⌋(SRS_Dlt_00034)

[SWS_Dlt_00485] ⌈The call to this this function should be forwarded to the PDU-Router function PduR_CancelTransmitRequest (see [2]). At this time the reason for cancelling should be provided. ⌋()

8.6.2.6 DltCom_SetInterfaceStatus

[SWS_Dlt_00265] ⌈

Service name:	DltCom_SetInterfaceStatus	

Syntax:	`Dlt_ReturnType DltCom_SetInterfaceStatus(`
	`  uint8[4] com_interface,`
	`  uint8 new_status`
	`)`	

Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	com_interface	To interpret as a name for a interface Possible values are "DCM" - Interface to the Dcm (Diagnostic Interface) For the Dlt communication module user defined values are allowed, e.g.: "SER1", "eth0"
	new_status	0 - OFF 1 - ON
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Dlt_ReturnType	DLT_E_OK - Succeded DLT_E_IF_BUSY - The interface is busy DLT_E_IF_NOT_AVAILABLE - The interface to change the state is not available
Description:	--	

⌋(SRS_Dlt_00034)

8.6.3 Expected Interfaces from Gpt

8.6.3.1 Gpt_EnableNotification

[SWS_Dlt_00513] ⌈

Service name:	Gpt_EnableNotification	
Syntax:	`void Gpt_EnableNotification( Gpt_ChannelType Channel )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (but not for the same timer channel)	
Parameters (in):	Channel	Numeric identifier of the GPT channel.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Enables the interrupt notification for a channel (relevant in normal mode).	

⌋ ()

8.6.3.2 Gpt_StartTimer

[SWS_Dlt_00514] ⌈

Service name:	Gpt_StartTimer	
Syntax:	`void Gpt_StartTimer( Gpt_ChannelType Channel, Gpt_ValueType Value )`	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant (but not for the same timer channel)	
Parameters (in):	Channel	Numeric identifier of the GPT channel.
	Value	Target time in number of ticks.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Starts a timer channel.	

⌋ ()

9 Sequence diagrams

9.1 Dlt initialization

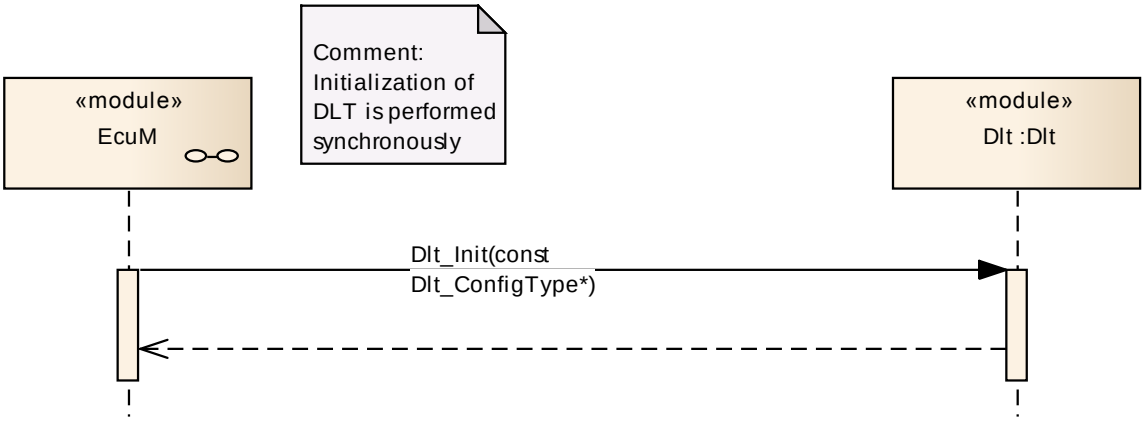


Figure 22: Sequence Dlt initialisation

The Preinit phase of Dlt is followed by the Init phase of Dlt.

9.2 General logging with Dlt

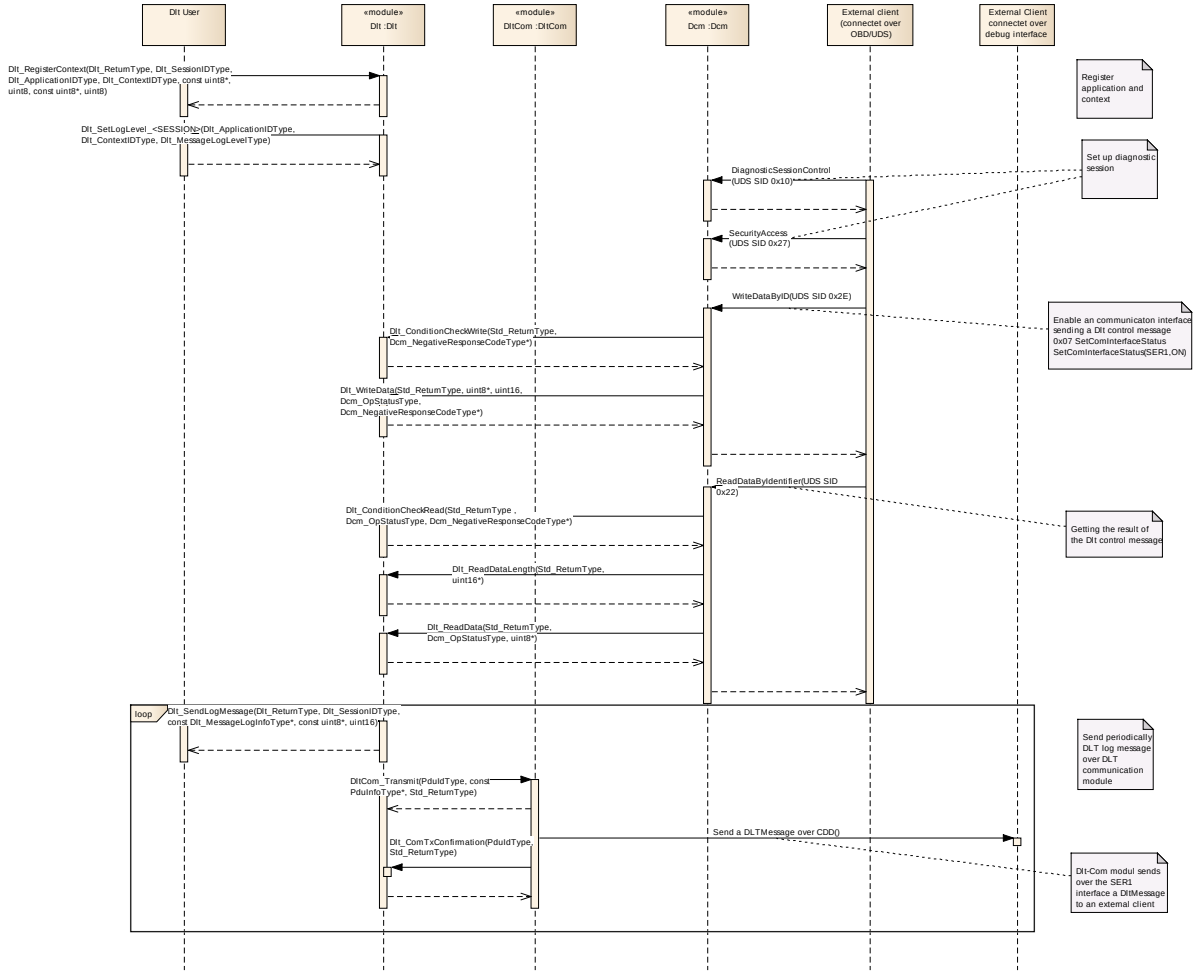


Figure 23: Sequence General logging with Dlt

This sequence chart corresponds to the Use Case described in 7.2.1.

9.3 Logging over UDS by using the Dcm interfaces

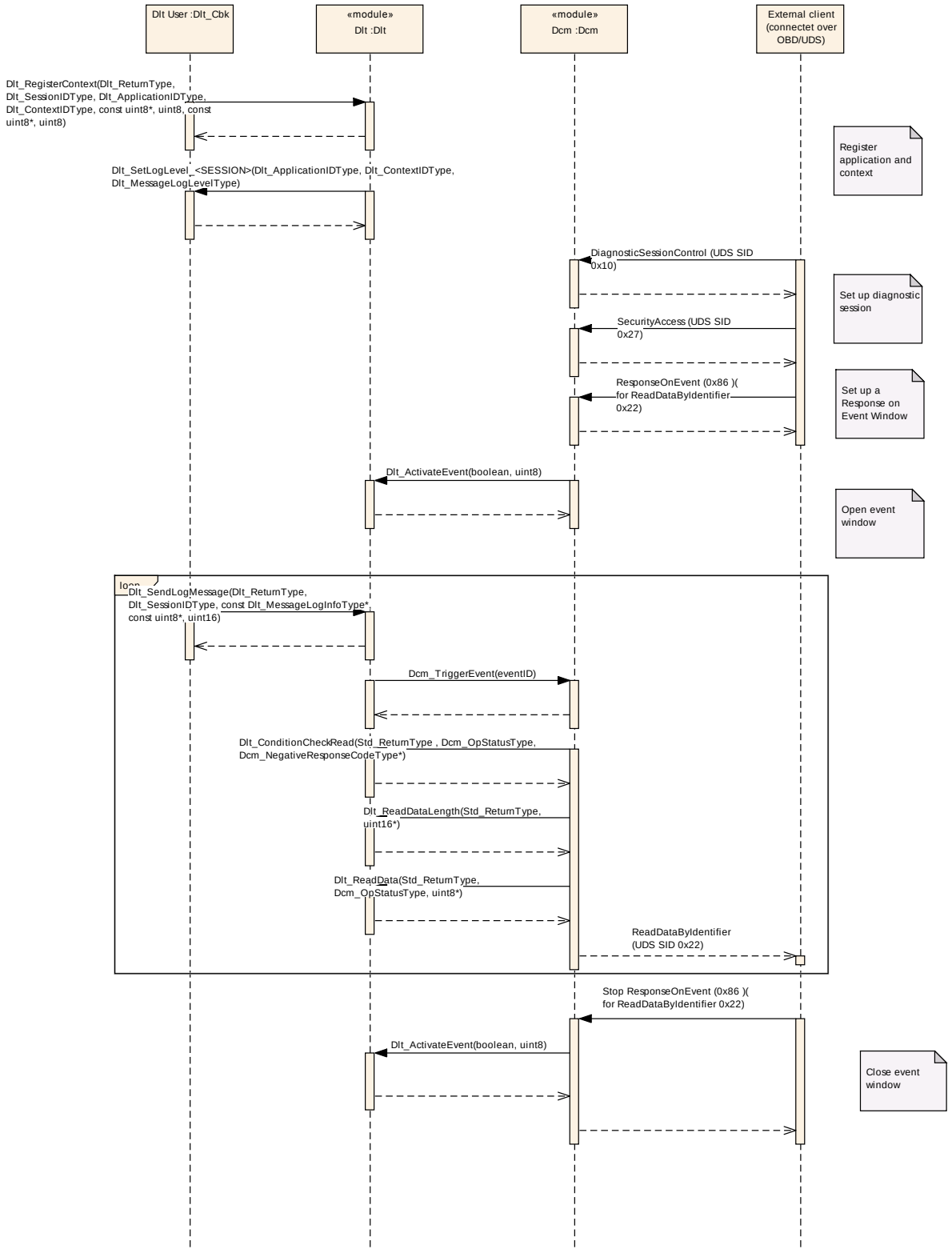


Figure 24: Sequence Logging over UDS with Dlt

This sequence chart corresponds to the Use Case described in 7.2.2.

9.4 Tracing of VFB

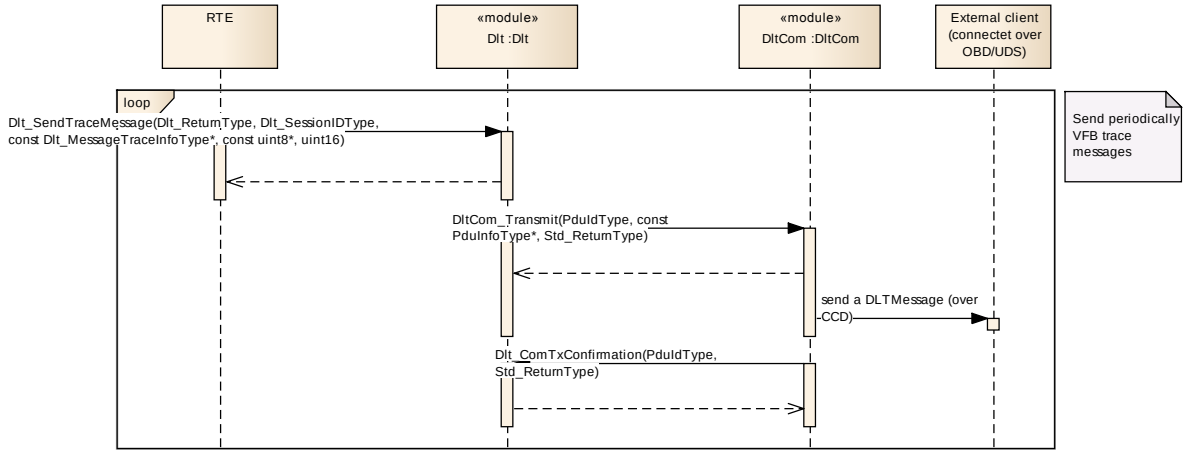


Figure 25: Sequence Tracing of VFB

This sequence chart corresponds to the Use Case described in 7.2.3.

9.5 Runtime configuration of Dlt

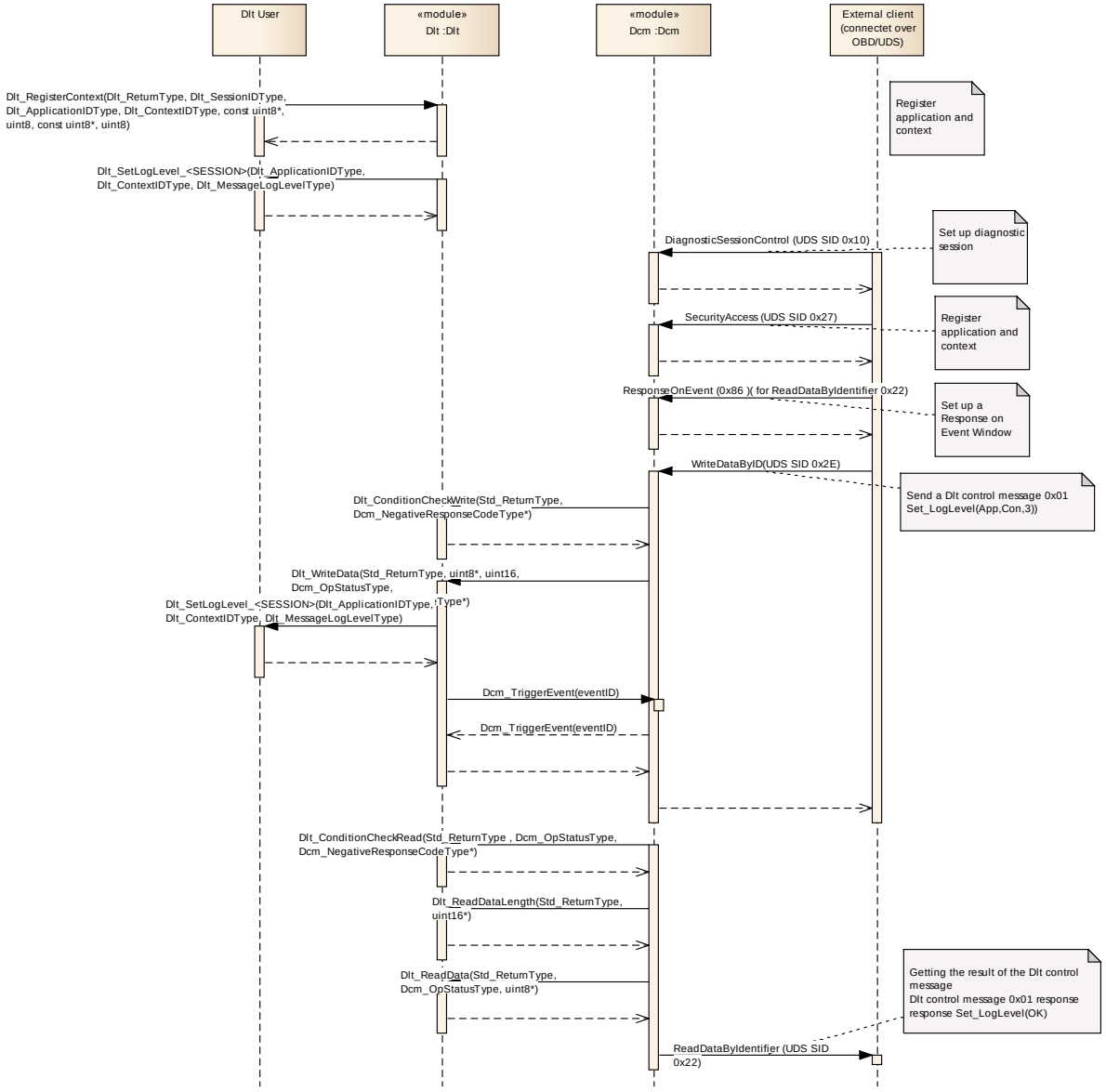


Figure 26: Sequence Runtime configuration of Dlt

This sequence chart corresponds to the Use Case described in 7.2.4.

9.6 Dlt interaction only over Dlt communication module

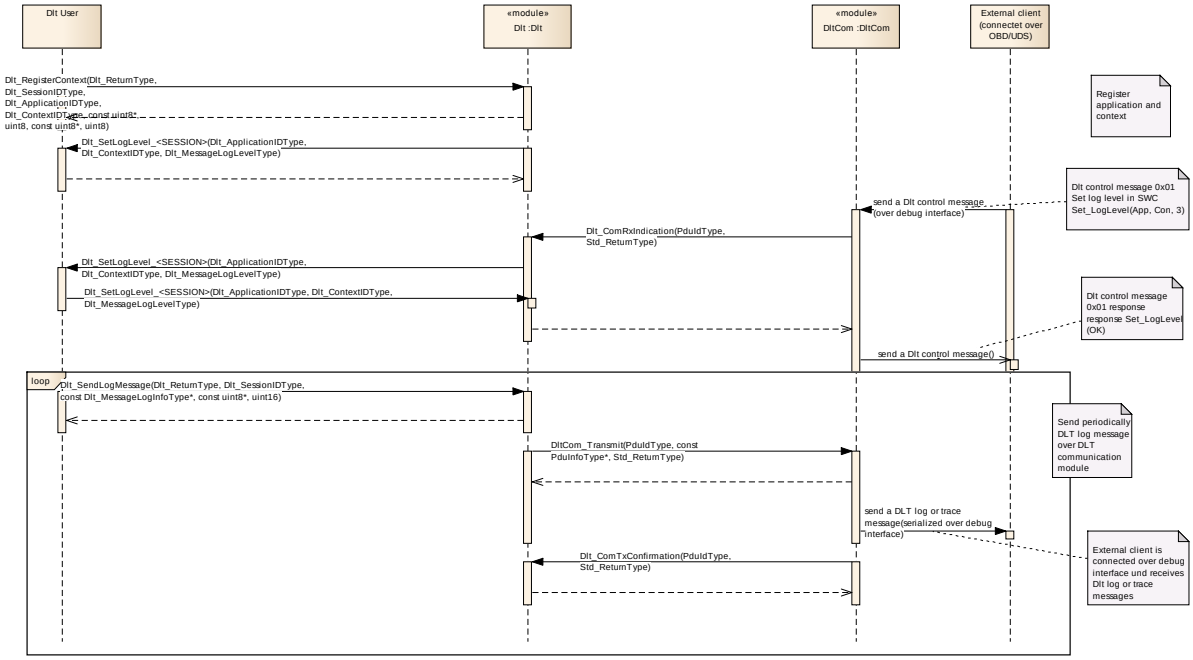


Figure 27: Sequence Dlt interaction only over Dlt communication module

This sequence chart corresponds to the Use Case described in 7.2.5.

9.7 Dlt interaction with Dem

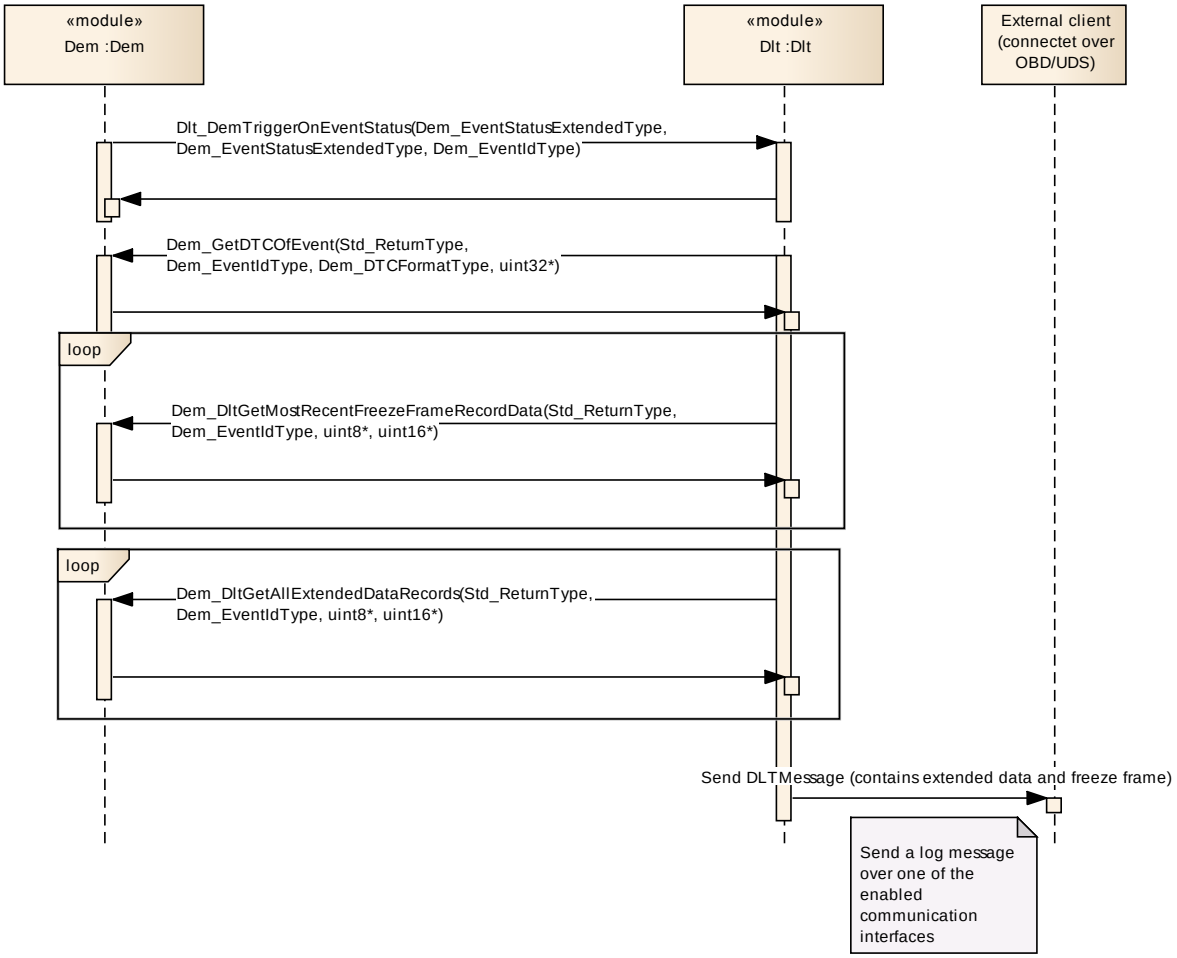
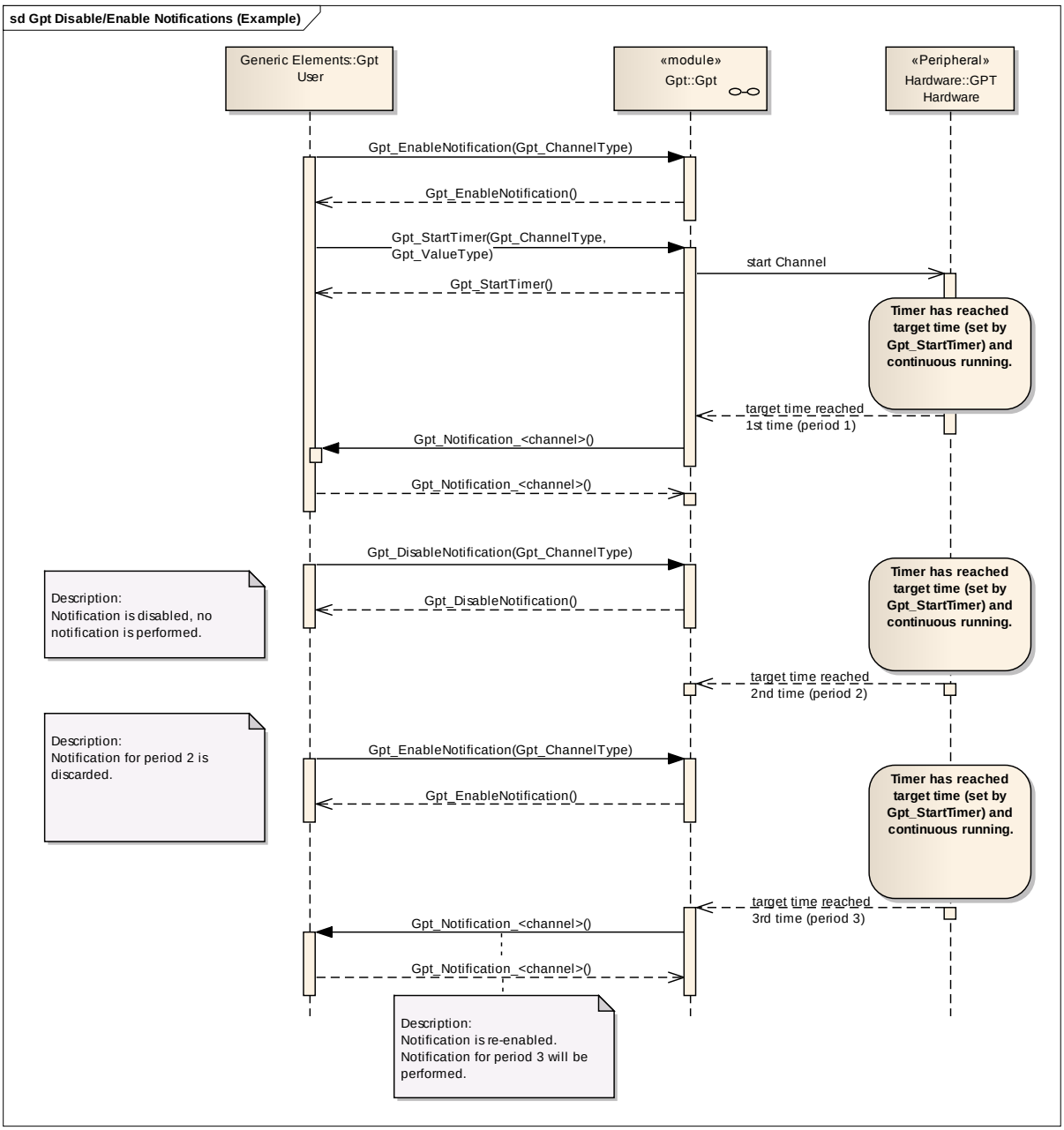


Figure 28 Interaction with Dem

Dlt get a trigger from Dem (Dlt_DemTriggerOnEventStatus) when an event in Dem changes. Than Dlt calls Dem for the DTC, the FreezeFrame and the ExtendedDataRecord of this event. Afterwards the collected information are send to an external client.

9.8 Dlt interaction with Gpt



10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module Dlt.

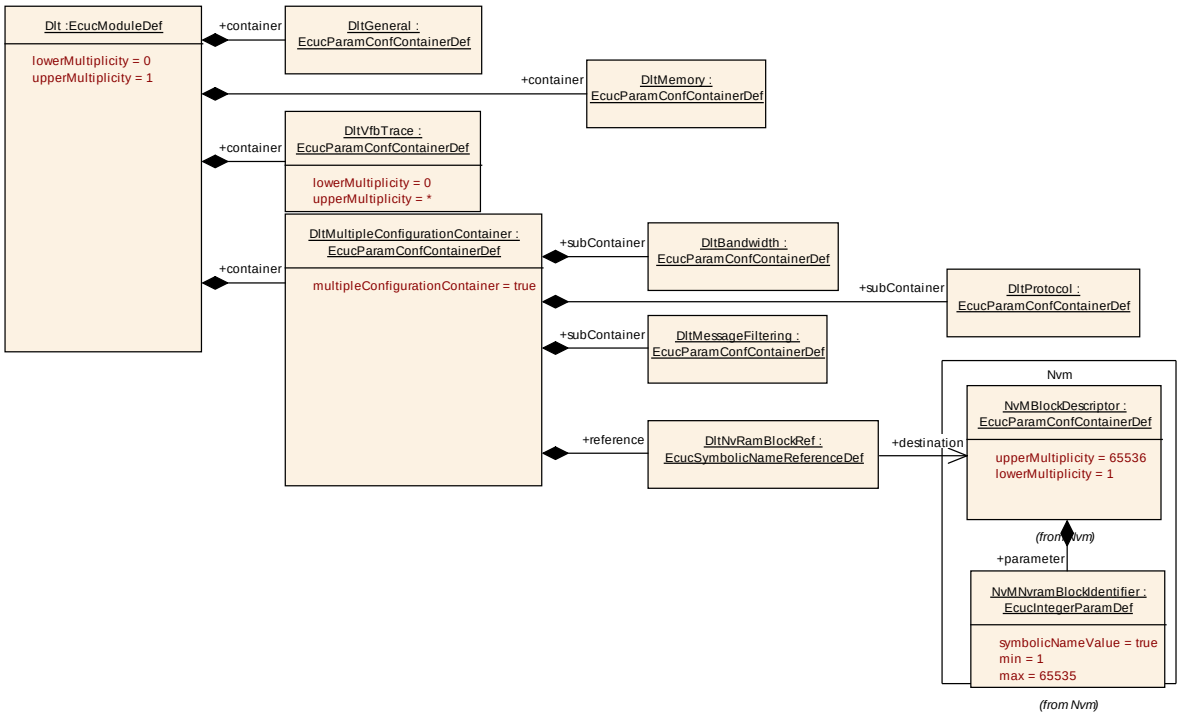
Chapter 10.3 specifies published information of the module <Module Name>.

10.1 How to read this chapter

.
For details refer to the chapter 10.1 “Introduction to configuration specification” in *SWS_BSWGeneral.*

10.2 Containers and configuration parameters

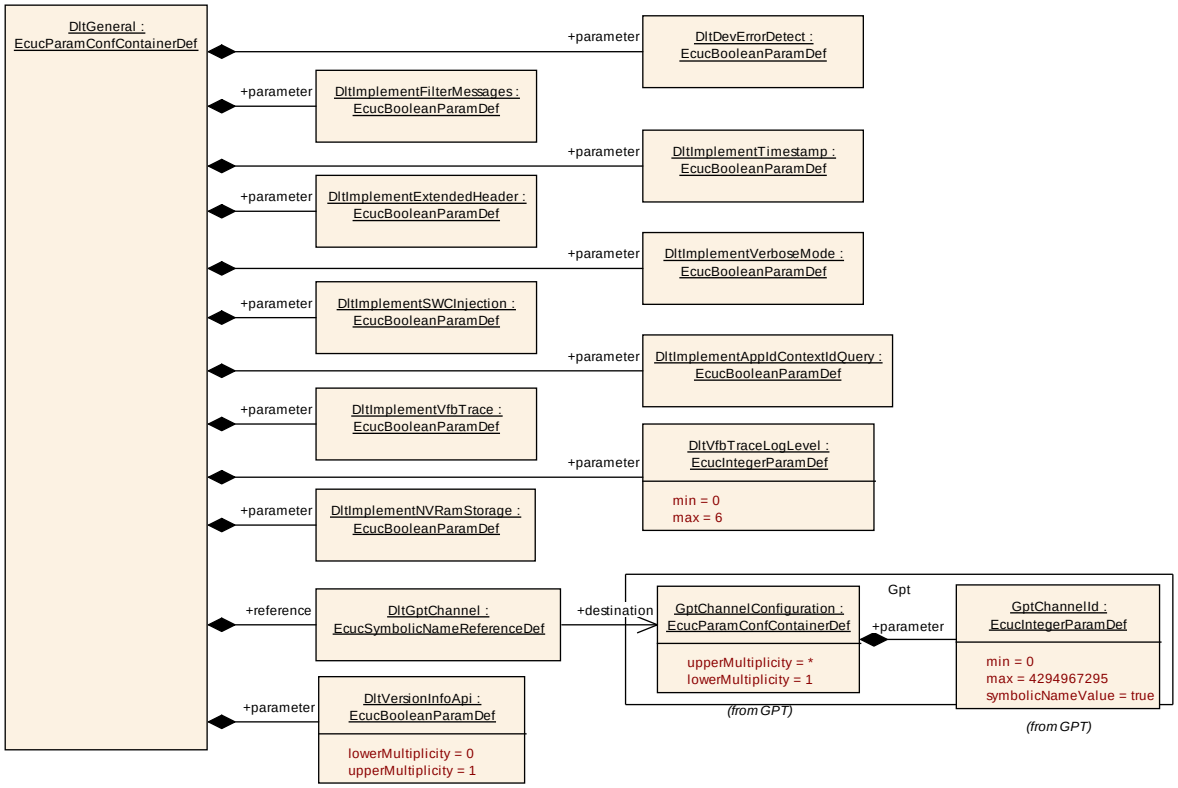
The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 0 and Chapter 8.



10.2.1 Dlt

SWS Item	**ECUC_Dlt_00800 :**
Module Name	*Dlt*
Module Description	--

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
DltGeneral	1	Flags for adding removing functionality from code.
DltMemory	1	Configuration parameters for reserving memory for some internal storing and buffer.
DltMultipleConfigurationContainer	1	Container holding the sub-structure for multiple configuration support.
DltVfbTrace	0..*	All functions to trace from the VFB by the Dlt.



10.2.2 DltGeneral

SWS Item	**ECUC_Dlt_00809 :**
Container Name	DltGeneral
Description	Flags for adding removing functionality from code.
Configuration Parameters	

SWS Item	**ECUC_Dlt_00840 :**		
Name	DltDevErrorDetect {DLT_DEV_ERROR_DETECT}		
Description	Enables/Disables development error detection.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dlt_00815 :**		
Name	DltImplementAppIdContextIdQuery		
Description	If set the functionality for Verbose Mode shall be available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dlt_00816 :**		
Name	DltImplementExtendedHeader		
Description	If set the extended functionality for the header shall be available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dlt_00817 :**		
Name	DltImplementFilterMessages		
Description	This flag is for code generation of Dlt. If set the functionality for filtering messages shall be included in the code.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dlt_00818 :**		
Name	DltImplementNVRamStorage		
Description	If set the functionality for storing and loading information in and from NVRam shall be available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dlt_00819 :**		
Name	DltImplementSWCInjection		
Description	If the remote call from functions over the Dlt in SW-C shall be available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dlt_00820 :**		
Name	DltImplementTimestamp		
Description	If set the timestamp functionality for the header shall be available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Dlt_00821 :		
Name	DltImplementVerboseMode		
Description	If set the functionality for Verbose Mode shall be available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Dlt_00822 :		
Name	DltImplementVfbTrace		
Description	If set the the header files and the implementation of VFB-trace shall be generated.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

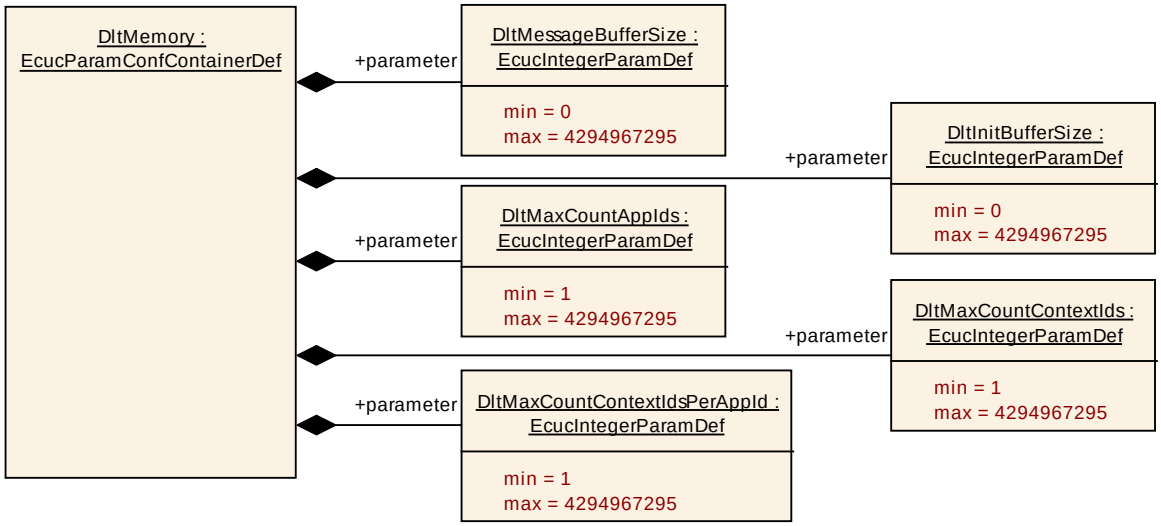
SWS Item	ECUC_Dlt_00844 :		
Name	DltVersionInfoApi		
Description	Pre-processor switch for enabling Version Info API support. True: version information API activated False: version information API deactivated		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Dlt_00839 :		
Name	DltVfbTraceLogLevel		
Description	The log level of the log messages generated by the VFB-Trace. ImplementationType: Dlt_MessageLogLevelType		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 6		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Dlt_00841 :
Name	DltGptChannel
Description	Reference to the hardware free running timer of the GPT module for time stamps (if no HWFRT is applied, calls to add timestamps are ignored).
Multiplicity	1

Type	Symbolic name reference to [GptChannelConfiguration]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.3 DltMemory

SWS Item	**ECUC_Dlt_00828 :**
Container Name	DltMemory
Description	Configuration parameters for reserving memory for some internal storing and buffer.
Configuration Parameters	

SWS Item	**ECUC_Dlt_00823 :**		
Name	DltInitBufferSize		
Description	Buffer size for the C-init buffer. This buffer is for storing messages from other BSW modules before Dlt is initialized. Unit: byte		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00824 :**
Name	DltMaxCountAppIds
Description	The maximum count of register able Application Ids. There shall be a table to manage registered Application IDs, this is the number of lines to hold in

	this table. Unit: byte		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

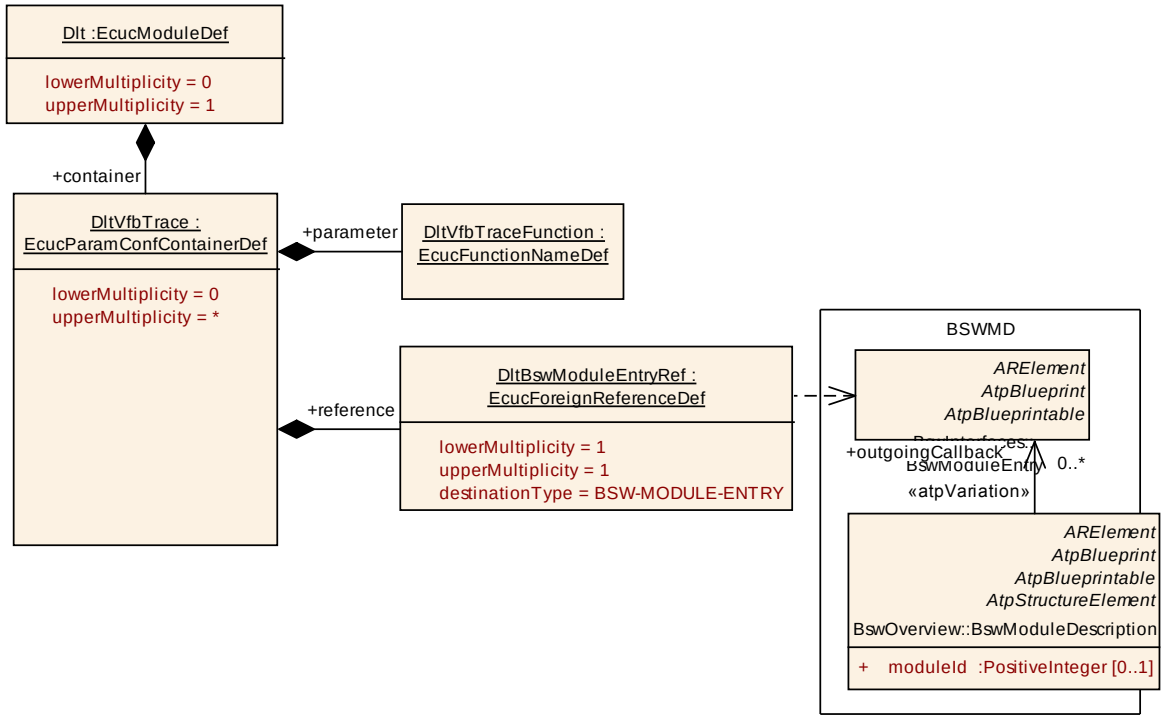
SWS Item	**ECUC_Dlt_00825 :**		
Name	DltMaxCountContextIds		
Description	The maximum count of registrable Context Ids. There shall be a table to manage registered Context IDs, this is the number of lines to hold in this table. Unit: byte		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00826 :**		
Name	DltMaxCountContextIdsPerAppId		
Description	Each Context ID belongs to a specific Application ID. Dlt shall handle the correlation between them. The table of the registered Application IDs shall hold for every Application ID several references to the proper Context IDs. This is the maximum count for Context IDs per Application ID. Unit: byte		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00829 :**		
Name	DltMessageBufferSize		
Description	Buffer size for storing Dlt messages for waiting to transmit over the Network (send buffer). Unit: byte		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD

	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers



10.2.4 DltVfbTrace

SWS Item	**ECUC_Dlt_00837 :**
Container Name	DltVfbTrace
Description	All functions to trace from the VFB by the Dlt.
Configuration Parameters	

SWS Item	**ECUC_Dlt_00838 :**		
Name	DltVfbTraceFunction		
Description	The Dlt generator shall enable VFB tracing for a given hook function when there is a #define in the Dlt-VFB configuration header file for the hook function name and tracing is globally enabled. Example: #define Dlt_Rte WriteHook i1 p1 a Start. Also the corresponding function shall be generated. The exact argument description for the function is to take from the provided BSWModudulDescription from the RTE module.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dlt_00804 :**		
Name	DltBswModuleEntryRef		
Description	Foreign reference to the BSWModuleEntry describing the trace function implementation.		
Multiplicity	1		
Type	Foreign reference to [BSW-MODULE-ENTRY]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

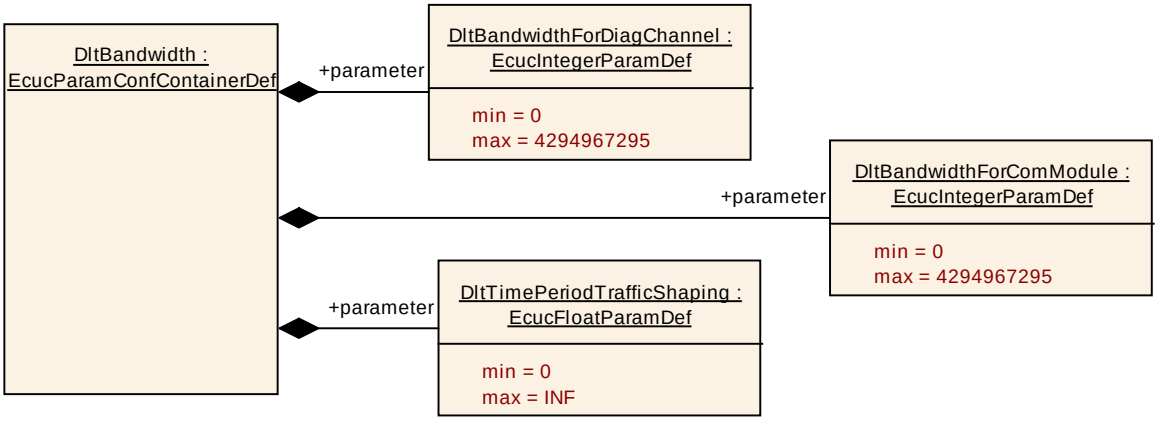
No Included Containers

10.2.5 DltMultipleConfigurationContainer

SWS Item	**ECUC_Dlt_00842 :**
Container Name	DltMultipleConfigurationContainer [Multi Config Container]
Description	Container holding the sub-structure for multiple configuration support.
Configuration Parameters	

SWS Item	**ECUC_Dlt_00831 :**		
Name	DltNvRamBlockRef		
Description	Reference to the NvM Block which is used to store the Dlt parameters.		
Multiplicity	1		
Type	Symbolic name reference to [NvMBlockDescriptor]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
DltBandwidth	1	Configuration parameters controlling network and diagnostic interfaces bandwidth. If DltImplementNVRamStorage is enabled this parameters are the initial values for the NVRam. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be uesd.
DltMessageFiltering	1	Configuration parameters for setting message filtering properties in Dlt module.
DltProtocol	1	Configuration parameters for handling the specific protocol variants.



10.2.6 DltBandwidth

SWS Item	**ECUC_Dlt_00801 :**
Container Name	DltBandwidth
Description	Configuration parameters controlling network and diagnostic interfaces bandwidth. If DltImplementNVRamStorage is enabled this parameters are the initial values for the NVRam. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be uesd.
Configuration Parameters	

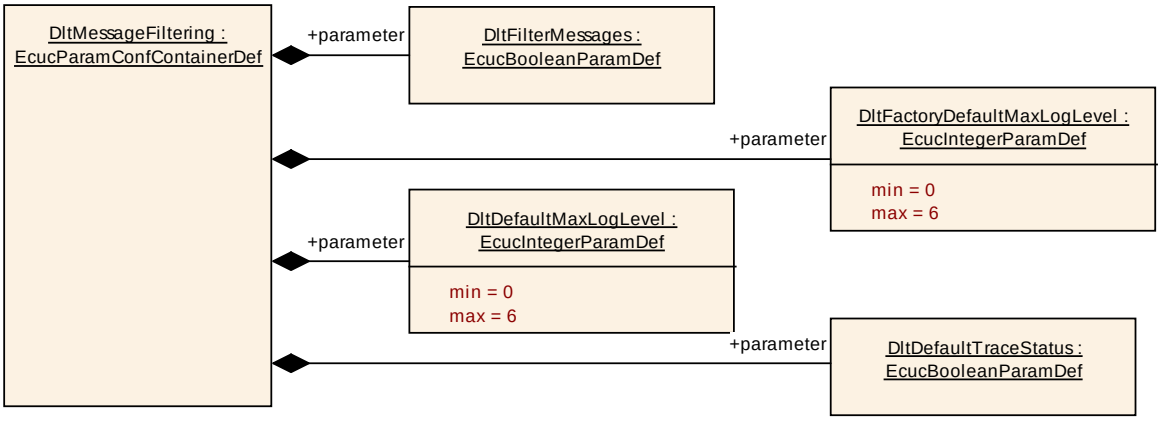
SWS Item	**ECUC_Dlt_00802 :**		
Name	DltBandwidthForComModule		
Description	For communication over the Dlt Communication Module the maximum bandwidth shall be set. Unit: kbit per second		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00803 :**		
Name	DltBandwidthForDiagChannel		
Description	For communication over the DCM and as follows over the diagnostic interface the maximum bandwidth shall be set. Unit: kbit per second		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00835 :**
Name	DltTimePeriodTrafficShaping

Description	For implementing a traffic shaping, a time window for measuring shall be provided. Unit: second		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers



10.2.7 DltMessageFiltering

SWS Item	**ECUC_Dlt_00830 :**
Container Name	DltMessageFiltering
Description	Configuration parameters for setting message filtering properties in Dlt module.
Configuration Parameters	

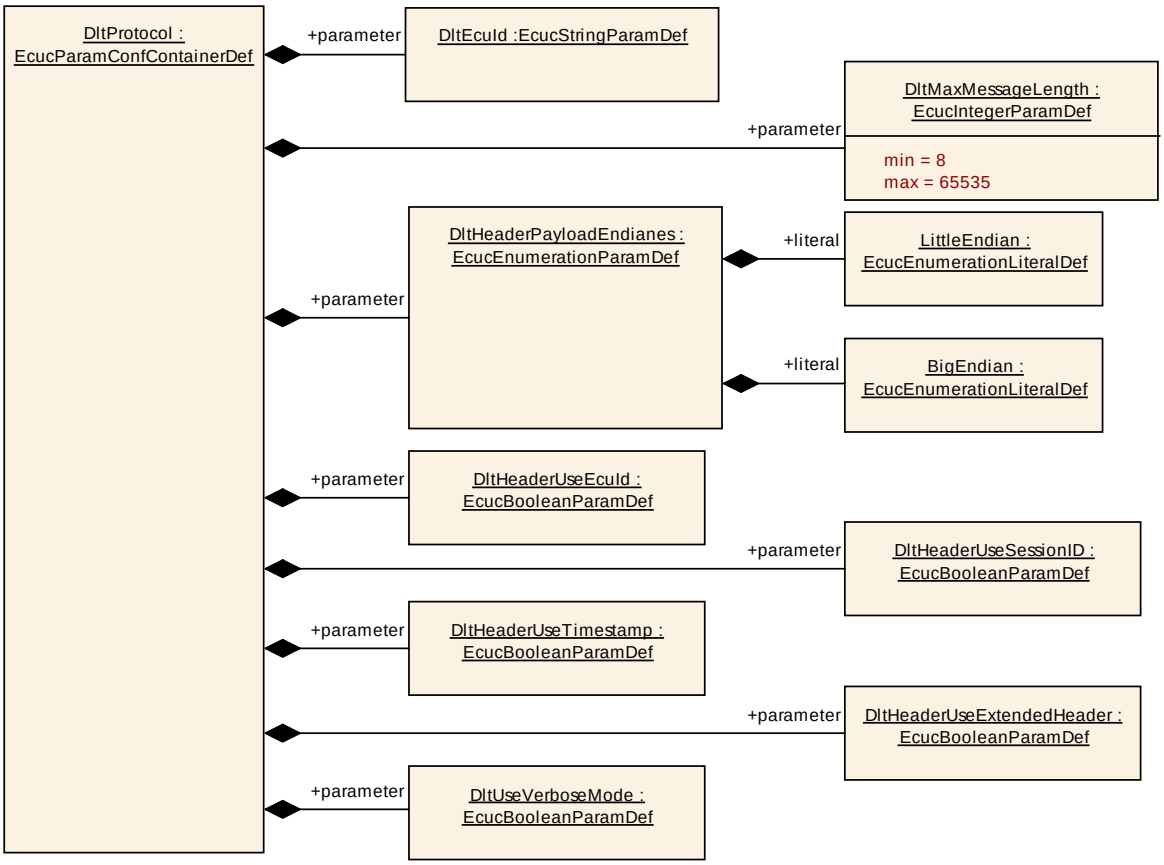
SWS Item	**ECUC_Dlt_00805 :**		
Name	DltDefaultMaxLogLevel		
Description	The maximum log level a received message (from SW-C to Dlt) can have. This can also be modified at runtime and stored persistently in NVRAM. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used. The value 0 means logging is disabled. ImplementationType: Dlt_MessageLogLevelType		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 6		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00843 :**		
Name	DltDefaultTraceStatus		
Description	Tells if trace messages shall be forwarded by Dlt. This functionality can also be modified at runtime and changed can stored persistently in NVRAM. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00807 :**		
Name	DltFactoryDefaultMaxLogLevel		
Description	The maximum log level a received message (from SW-C to Dlt) can have. This is for setting DltDefaultMaxLogLevel to factory defaults. The value 0 means logging is disabled. ImplementationType: Dlt_MessageLogLevelType		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 6		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00808 :**		
Name	DltFilterMessages		
Description	This flag gives the Dlt the instruction to filter or not to filter incoming log or trace messages. If it is not set all incoming messages are forwarded to the communication channel. So also the caller of the DltSendXXXMessage can leave the field trace_info or log_info empty. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Can only be true if DltImplementFilterMessages is true.		

No Included Containers



10.2.8 DltProtocol

SWS Item	**ECUC_Dlt_00832 :**
Container Name	DltProtocol
Description	Configuration parameters for handling the specific protocol variants.
Configuration Parameters	

SWS Item	**ECUC_Dlt_00806 :**		
Name	DltEcuId		
Description	This is the name of the ECU for use within the Dlt protocol. The meaning is described in the document. This name is transmitted within the Dlt protocol. There this are 4 characters. If you want to use an number representation type this as character.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00810 :**

Name	DltHeaderPayloadEndianes		
Description	Determines the endianess of the CPU (Most Significant Byte).		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	BigEndian	--	
	LittleEndian	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00811 :**		
Name	DltHeaderUseEcuId		
Description	Corresponds to field WEID (With ECU ID). If set ECU ID shall be placed in the header, else not. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00812 :**		
Name	DltHeaderUseExtendedHeader		
Description	Corresponds to field UEH (Use Extended Header). If set the Extended Header shall be attached, else not. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: Can only be true if DltImplementExtendedHeader is true.		

SWS Item	**ECUC_Dlt_00813 :**		
Name	DltHeaderUseSessionID		
Description	Corresponds to field WSID (with Session ID). If set the Session ID shall be placed in the header, else not. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00814 :**		
Name	DltHeaderUseTimestamp		
Description	Corresponds to field WTMS (With Timestamp). If set the timestamp shall be placed in the header, else not. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: Can only be true if DltImplementTimestamp is true.		

SWS Item	**ECUC_Dlt_00827 :**		
Name	DltMaxMessageLength		
Description	The maximum length of a Dlt log or trace message.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	8 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Dlt_00836 :**		
Name	DltUseVerboseMode		
Description	If this flag is set Dlt shall use the Verbose Mode for generating the header of the transport protocol. Also it shall store the information provided by registering Context ID and Application ID (description) at runtime if flag is set. If it is not set, the Non Verbose Mode shall be used. If DltImplementNVRamStorage is enabled this parameter is the initial value for the corresponding NVRam entry. If DltImplementNVRamStorage is not set, Link-Time or Post-Build configuration shall be used.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: Can only be true if DltImplementVerboseMode is true.		

No Included Containers

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in *SWS_BSWGeneral.*

11 Not applicable requirements

[SWS_Dlt_00511] ⌈These requirements are not applicable to this specification.⌋ (SRS_BSW_00170, SRS_BSW_00387, SRS_BSW_00395, SRS_BSW_00400, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00437, SRS_BSW_00168, SRS_BSW_00427, BSW00431, SRS_BSW_00432, BSW00434, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00409, SRS_BSW_00386, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00326, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00160, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00373, SRS_BSW_00314, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00439, SRS_BSW_00376)