Document Title	Log and Trace Protocol Specification
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	787

Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R19-11

Document Change History

Date	Release	Changed by	Description
2019-11-28	R19-11	AUTOSAR Release Management	• Added Proposal Usage of LogLevels • Added Recommendation to transmit IDs of arbitrary length • Editorial changes • Changed Document Status from Final to published
2019-03-29	1.5.1	AUTOSAR Release Management	• No content changes
2018-10-31	1.5.0	AUTOSAR Release Management	• LT Command SyncTimeStamp added • Editorial changes
2018-03-29	1.4.0	AUTOSAR Release Management	• No content changes
2017-12-08	1.3.0	AUTOSAR Release Management	• No content changes
2017-10-27	1.2.0	AUTOSAR Release Management	• Enhanced formal quality of requirements and requirements tracing
2017-03-31	1.1.0	AUTOSAR Release Management	• Added requirement for the header format (Big-endian)
2016-11-30	1.0.0	AUTOSAR Release Management	• Initial Release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

- AUTOSAR confidential -

1 Introduction and overview

This protocol specification specifies the format, message sequences and semantics of the AUTOSAR Protocol Dlt.

The protocol allows sending Diagnostic, Log and Trace information onto the communications bus.

Therefore, the Dlt module collects debug information from applications or other software modules, adds metadata to the debug information, and sends it to the communications bus.

In addition, the Dlt Protocol allows filtering debug information depending on the severity level, e.g.: “fatal error” or “information”. This filter can be modified during runtime via Dlt Control Messages sent by an external Logging Tool.

It is also possible to directly inform applications about the new filter level to only generate debug information especially for this selected severity level, assign messages to another communications bus at runtime, or to store the modified Dlt configuration nonvolatile (if supported by hardware).

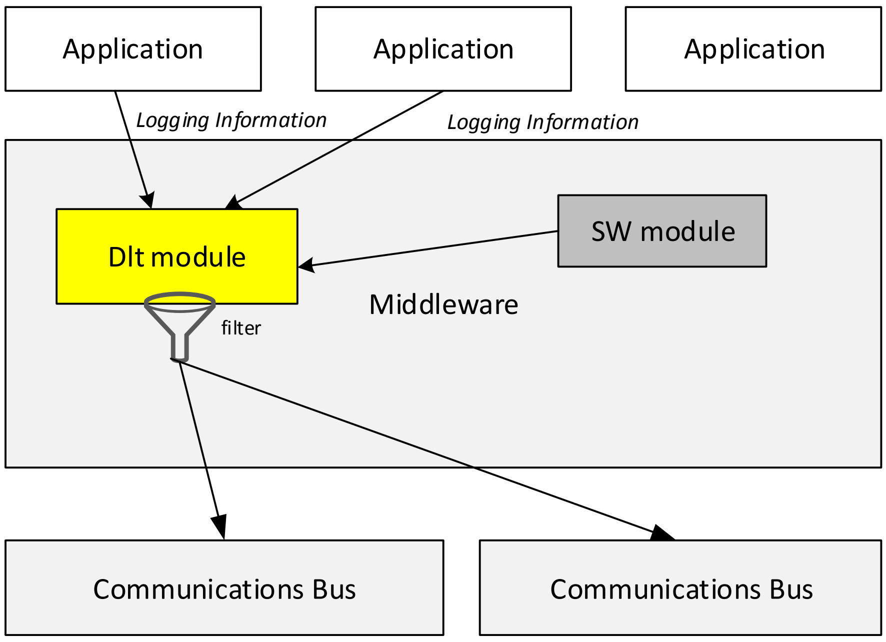


Figure 2 – Location of the Dlt Protocol

1.1 Purpose

The Dlt protocol can be used at the ECU’ development phase to log and store debug information externally on a logging device.

1.2 Applicability of the protocol

It is intended to use the Dlt Protocol at the development phase of an ECU. It is assumed to use an external logging- and tracing tool to store the debug information generated by the ECU.

This logging- and tracing tool is also needed to modify the filter setting at runtime if wanted, or to store the current Dlt configuration of the ECU persistently.

1.2.1 Safety and security considerations

It is highly recommended to deactivate the Dlt functionality after the development phase is over. In particular, the Injection-Feature should be deactivated in any case!

The activation and deactivation of the Dlt functionality should be done using a security mechanism.

1.2.2 Constraints and assumptions

The Dlt Protocol is designed to work “connectionless”. This means that no external communication or other stimulation is needed to use the Dlt protocol.

Although there is no need to connect an external Logging tool, it makes sense having one, which stores and interprets the received debug messages. This device can also be used to generate Dlt Control Messages to influence the ECU, like modifying the filter setting (i.e. change the severity level of the debug information).

1.2.3 Limitations

The available (free) bandwidth of the communications bus should be taken into consideration to not influence the regular communication too much.

1.3 Dependencies to other protocol layers

[1] IEC 7498-1 The Basic Model, IEC Norm, 1994

[2] ASAM MCD-2 NET (FIBEX)

1.4 Dependencies to the Application Layer

To transmit Dlt messages, the applications need to know whether to send the Dlt messages using the verbos- or non-verbose mode.

In addition, the applications may offer the possibility to get informed about a filter setting change. For this purpose, the applications should register themselves at the Dlt module.

2 Use Cases

This chapter describes the use cases which can be realized by an environment of an ECU which implements the Dlt Protocol.

Although the Dlt protocol is bus agnostic, it is recommended to use communications busses with higher bandwidth like Ethernet. Nonetheless it is not limited to it.

2.1.1 Use Case general logging with Dlt

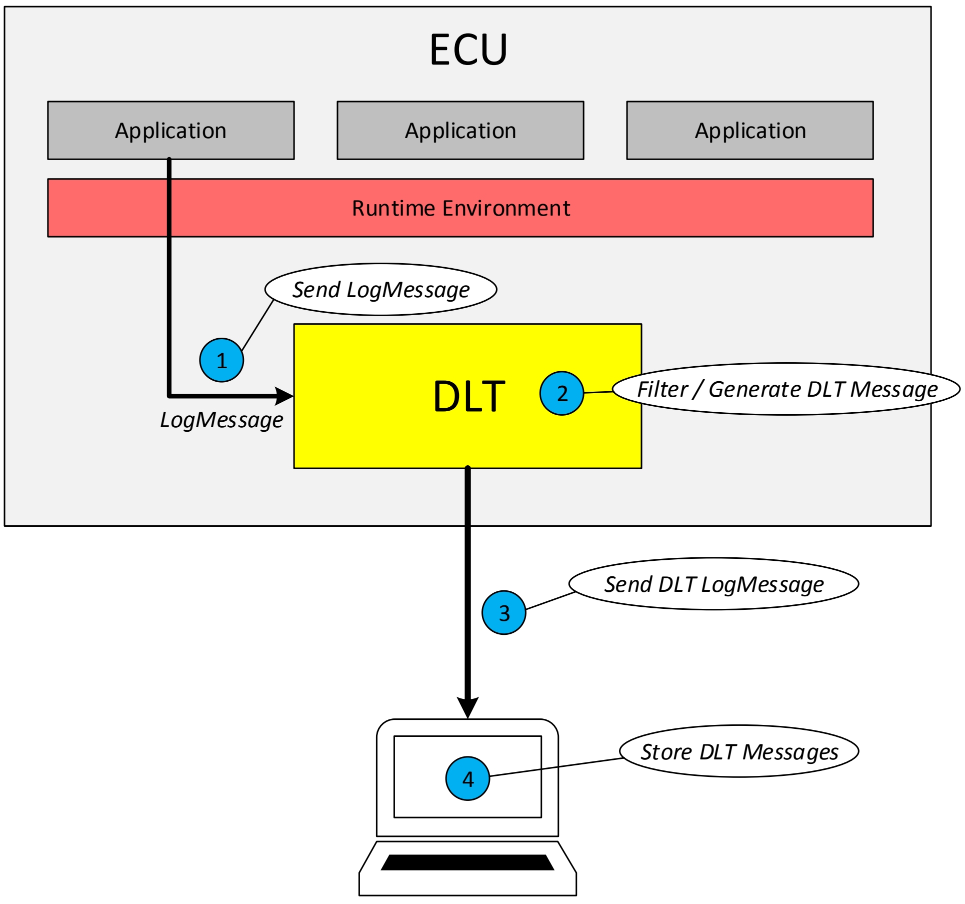


Figure 3: General logging with Dlt

(1) An application / SW-C is generating a Log Message.
(2) The Log Message is sent to a Dlt module, which implements the Dlt Protocol
(3) The Dlt module sends the log message to the communications bus
(4) An external Dlt client records the log message

2.1.2 Use Case tracing of VFB

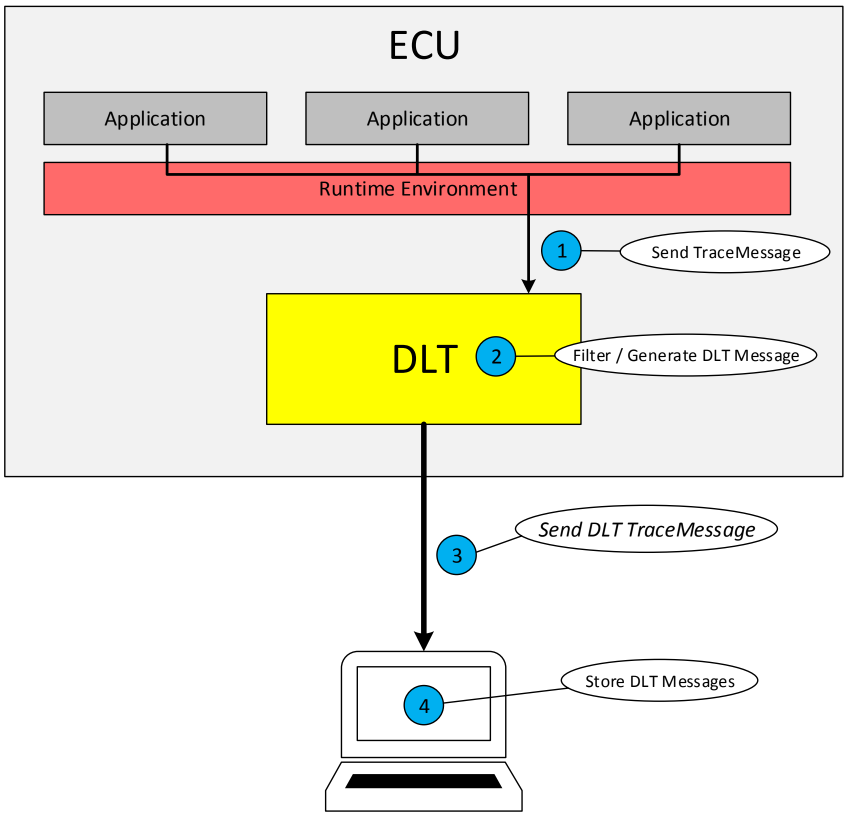


Figure 4: Tracing of VFB

(1) RTE calls the macro provided by Dlt, which calls the Dlt API generating the trace message.
(2) The Dlt module which implements the Dlt Protocol sends the trace message to the implemented Dlt communication module interface.
(3) The Dlt communication module forwards the trace message to the network.
(4) An external client receives and stores the trace messages.

2.1.3 Use Case runtime configuration of Dlt

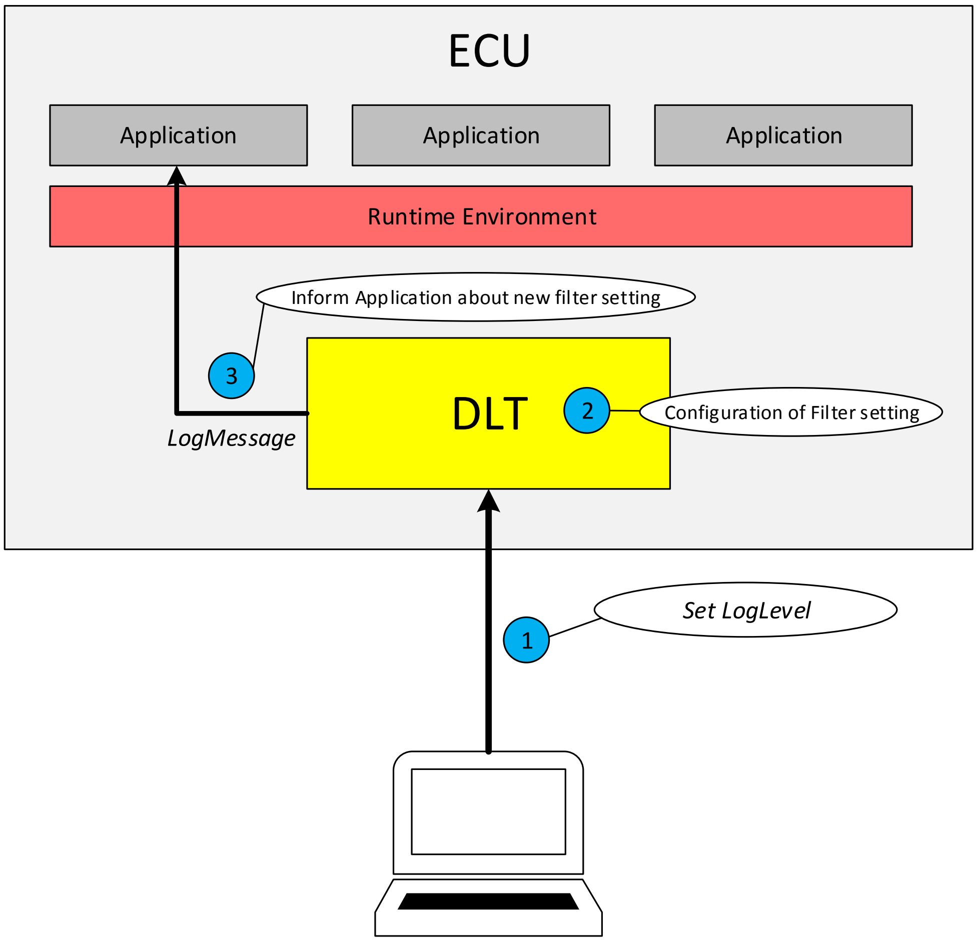


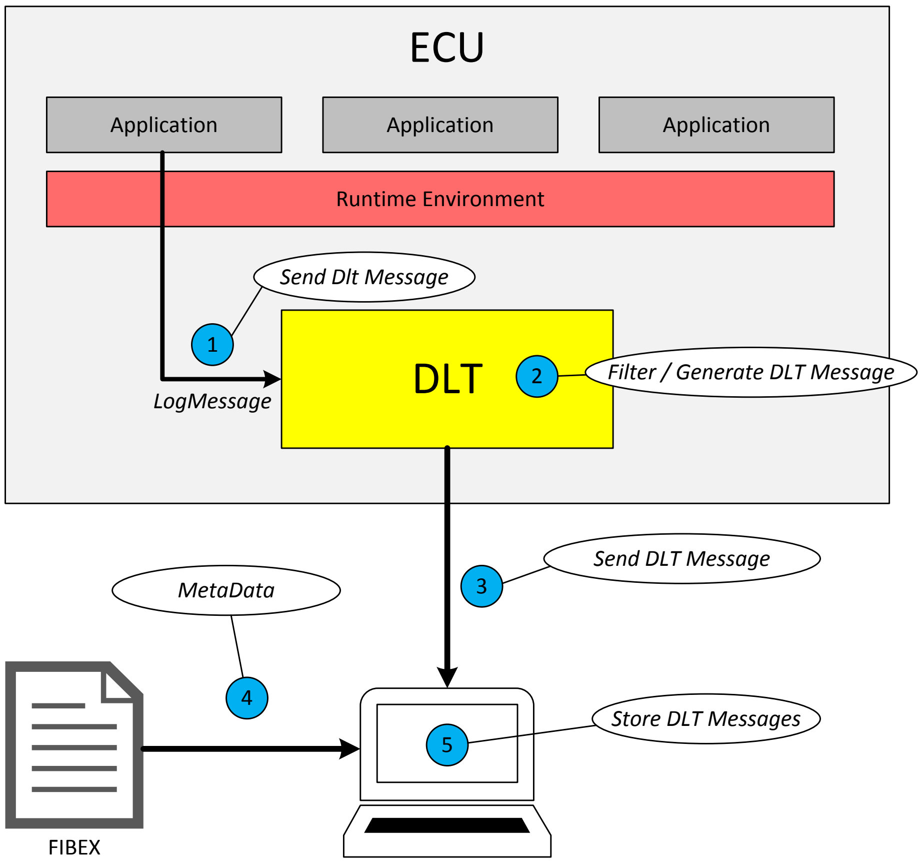
Figure 5: Runtime configuration of Dlt

(1) An external Dlt client sets the log and trace level and sends the change to the Dlt module, which implements the Dlt Protocol.
(2) The Dlt module adapts its configuration of filter settings accordingly.
(3) The Dlt module informs the application about the new log level.

2.1.4 Use Case non-verbose mode

To reduce the amount of traffic on the bus, it can be avoided to send meta data about variables on the communications bus.

Instead, an external FIBEX file holds the information how the payload shall be interpreted. The external Dlt client merges and stores these meta data with the received parameter values.



(1) The Dlt module is called to transmit a Dlt message in non-verbose mode.
(2) The Dlt module filters and generate the Dlt message
(3) The Dlt module sends the Dlt message to the communications bus.
(4) An external Dlt client fetches meta information from an external FIBEX file.
(5) The merged information is stored by an external Dlt client.

3 Requirements traceability

Requirement	Description	Satisfied by
RS_LT_00002	All log and trace messages sent by an ECU shall have a standardized transmission format and a standardized storage format.	PRS_Dlt_00094, PRS_Dlt_00120, PRS_Dlt_00126, PRS_Dlt_00135, PRS_Dlt_00292, PRS_Dlt_00320, PRS_Dlt_00324, PRS_Dlt_00325, PRS_Dlt_00326, PRS_Dlt_00404, PRS_Dlt_00405, PRS_Dlt_00427, PRS_Dlt_00458, PRS_Dlt_00600, PRS_Dlt_00601, PRS_Dlt_00602, PRS_Dlt_00603, PRS_Dlt_00604, PRS_Dlt_00605, PRS_Dlt_00606, PRS_Dlt_00607, PRS_Dlt_00608, PRS_Dlt_00609, PRS_Dlt_00610, PRS_Dlt_00611, PRS_Dlt_00612, PRS_Dlt_00614, PRS_Dlt_00617, PRS_Dlt_00618, PRS_Dlt_00619, PRS_Dlt_00620, PRS_Dlt_00621, PRS_Dlt_00622, PRS_Dlt_00623, PRS_Dlt_00641
RS_LT_00013	The transmitted data shall be packetized.	PRS_Dlt_00091, PRS_Dlt_00314, PRS_Dlt_00315, PRS_Dlt_00409
RS_LT_00014	The transport format shall be binary.	PRS_Dlt_00378
RS_LT_00016	The format shall deal with Big and Little Endianess.	PRS_Dlt_00091, PRS_Dlt_00604, PRS_Dlt_00605
RS_LT_00017	Each log and trace message shall contain a timestamp, which will be added to the message during reception of the message in the LT module.	PRS_Dlt_00112, PRS_Dlt_00309, PRS_Dlt_00323, PRS_Dlt_00628
RS_LT_00018	A global message counter shall be implemented, to detect messages loss.	PRS_Dlt_00105, PRS_Dlt_00106, PRS_Dlt_00319, PRS_Dlt_00613
RS_LT_00020	The log and trace message shall contain a parameter, which represents the source of the log and trace message.	PRS_Dlt_00322, PRS_Dlt_00616
RS_LT_00021	There shall be a logical grouping for log messages by using different identifiers.	PRS_Dlt_00127, PRS_Dlt_00128, PRS_Dlt_00312, PRS_Dlt_00313
RS_LT_00022	Each ECU shall have its unique ECU ID.	PRS_Dlt_00308, PRS_Dlt_00321, PRS_Dlt_00606, PRS_Dlt_00607, PRS_Dlt_00615
RS_LT_00023	The payload shall transport the parameters of a log and trace message.	PRS_Dlt_00134, PRS_Dlt_00314, PRS_Dlt_00315, PRS_Dlt_00353, PRS_Dlt_00378, PRS_Dlt_00409, PRS_Dlt_00459, PRS_Dlt_00623
RS_LT_00024	It shall be possible to transmit the parameters in a raw format.	PRS_Dlt_00126, PRS_Dlt_00310
RS_LT_00025	It shall be possible to transmit ASCII text in log or trace messages.	PRS_Dlt_00155, PRS_Dlt_00156, PRS_Dlt_00157, PRS_Dlt_00182, PRS_Dlt_00183, PRS_Dlt_00366, PRS_Dlt_00367, PRS_Dlt_00373, PRS_Dlt_00392, PRS_Dlt_00400, PRS_Dlt_00410, PRS_Dlt_00411, PRS_Dlt_00420, PRS_Dlt_00627

RS_LT_00026	The data in non-verbose mode shall be described by an extra file.	PRS_Dlt_00134, PRS_Dlt_00353, PRS_Dlt_00396, PRS_Dlt_00397, PRS_Dlt_00398, PRS_Dlt_00399, PRS_Dlt_00401, PRS_Dlt_00424, PRS_Dlt_00425
RS_LT_00027	Each message in non-verbose mode shall have a unique Message ID significant for identifying the source of the tracing.	PRS_Dlt_00352, PRS_Dlt_00397, PRS_Dlt_00398, PRS_Dlt_00624
RS_LT_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.	PRS_Dlt_00194, PRS_Dlt_00195, PRS_Dlt_00196, PRS_Dlt_00197, PRS_Dlt_00198, PRS_Dlt_00199, PRS_Dlt_00200, PRS_Dlt_00380, PRS_Dlt_00381, PRS_Dlt_00383, PRS_Dlt_00393, PRS_Dlt_00494, PRS_Dlt_00502, PRS_Dlt_00635, PRS_Dlt_00637, PRS_Dlt_00638, PRS_Dlt_00639, PRS_Dlt_00640
RS_LT_00033	A list of all log and trace sources of an ECU shall be accessible from the external client.	PRS_Dlt_00197
RS_LT_00037	There shall be a buffer to store log and trace message locally.	PRS_Dlt_00769, PRS_Dlt_00770
RS_LT_00039	The LT shall provide the possibility to store configuration data in a persistent way.	PRS_Dlt_00199
RS_LT_00040	The LT component shall be able to filter log and trace messages.	PRS_Dlt_00205
RS_LT_00044	Provide raw buffer content.	PRS_Dlt_00119, PRS_Dlt_00135, PRS_Dlt_00139, PRS_Dlt_00145, PRS_Dlt_00147, PRS_Dlt_00148, PRS_Dlt_00149, PRS_Dlt_00150, PRS_Dlt_00152, PRS_Dlt_00153, PRS_Dlt_00160, PRS_Dlt_00161, PRS_Dlt_00169, PRS_Dlt_00170, PRS_Dlt_00171, PRS_Dlt_00172, PRS_Dlt_00173, PRS_Dlt_00175, PRS_Dlt_00176, PRS_Dlt_00177, PRS_Dlt_00182, PRS_Dlt_00183, PRS_Dlt_00354, PRS_Dlt_00355, PRS_Dlt_00356, PRS_Dlt_00357, PRS_Dlt_00358, PRS_Dlt_00362, PRS_Dlt_00363, PRS_Dlt_00364, PRS_Dlt_00366, PRS_Dlt_00367, PRS_Dlt_00369, PRS_Dlt_00370, PRS_Dlt_00371, PRS_Dlt_00372, PRS_Dlt_00374, PRS_Dlt_00375, PRS_Dlt_00385, PRS_Dlt_00386, PRS_Dlt_00387, PRS_Dlt_00388, PRS_Dlt_00389, PRS_Dlt_00390, PRS_Dlt_00411, PRS_Dlt_00412, PRS_Dlt_00414, PRS_Dlt_00422, PRS_Dlt_00423, PRS_Dlt_00459, PRS_Dlt_00625, PRS_Dlt_00626, PRS_Dlt_00627

4 Acronyms and abbreviations

Abbreviation / Acronym	***Description***
APID	Application ID
CTID	Context ID
Dlt	Diagnostic Log and Trace
FIBEX	Field Bus Exchange Format
MCNT	Message Counter
MSBF	Most Significant Byte First
MSBI	Message Bus Info
MSCI	Message Control Info
MSLI	Message Log Info
MSTP	Message Type
MSTI	Message Trace Info
NOAR	Number of Arguments
STMS	Timestamp
UEH	Use Extended Header
VERB	Verbose
VERS	Version Number
WEID	With ECU ID
WSID	With Session ID
WTMS	With Timestamp

- AUTOSAR confidential -

4.1 Term and definition

Term	***Description:***
Log and trace message	A log and trace message contains all data and options to describe a log and trace event in a software. The log and trace message consists of a header and payload.
Dlt User	A Dlt User represents the source of a generated Dlt message. Possible users are Applications, RTE or other software modules
Log Message	A Log Message contains debug information like state changes or value changes.
Trace Message	A Trace messages contains information, which has passed via the VFB.
ECU ID	The ECU ID is the name of an ECU, composed by four 8-bit ASCII characters (e.g. ABS0 or COMB).
FIBEX	The Field Bus Exchange Format is a versatile XML bases description format.
Session	A session is the logical entity of the source of log or trace messages. If a SW-C / application is instantiated several times, a session for each instance with a globally unique session ID is used.
Session ID	The Session ID is the identification number of a log or trace session
Application ID	Application ID is an abbreviation of a SW-C / application. It identifies the SW-C / application the log and trace message originates.
Context ID	Context ID is a user defined ID to group Log and Trace Messages generated by a SW-C / application. The following rules apply: • Each Application ID can own several Context IDs. • Context IDs are grouped by Application IDs. • Context IDs shall be unique within an Application ID. • The source of a log and trace message is identified using the tuple “Application ID” and “Context ID”. Four 8-bit ASCII characters compose the Context ID.
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.
Log level	A log level defines a classification for the severity grade of a Log Message.
Trace status	The trace status provides information if a trace message should be send.
Log Channel	A physical Communications Bus which is used to transmit Dlt messages.
External client	The external client is a tool to control, monitor and store the log/trace information provided by the ECUs using the Dlt module.

5 Protocol specification

5.1 Message format

For both, debug data and control information, the same Dlt message format is used. It consists of a Standard Header, an optional Extended Header, and a Payload segment.

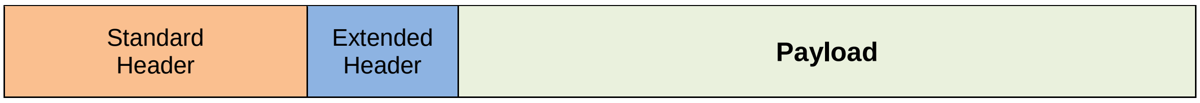
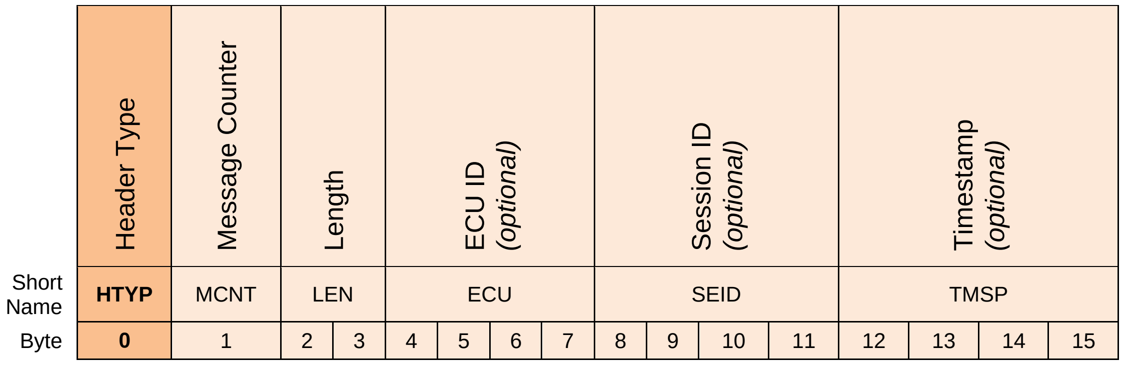


Figure 6 - Dlt message format

5.1.1 Header format

[PRS_Dlt_00091] ⌈The Standard Header and the Extended Header shall be in big endian format (MSB first).⌋ (RS_LT_00016,RS_LT_00013)

5.1.1.1 Standard Header



	Header Type	Message Counter	Length		ECU ID *(optional)*				Session ID *(optional)*				Timestamp *(optional)*			
Short Name	**HTYP**	MCNT	LEN		ECU				SEID				TMSP			
Byte	**0**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 7 - Dlt Standard Header

[PRS_Dlt_00458] ⌈The Dlt Standard Header shall consist of the following fields in the following order:

- Byte 0: HTYP (Header Type)
- Byte 1: MCNT (Message Counter)
- Byte 2-3: LEN (Length)
- Byte 4-7: ECU (ECU ID)
- Byte 8-11: SEID (Session ID)
- Byte 12-15: TMSP (Timestamp)

⌋ (RS_LT_00002)

5.1.1.1.1 Header Type

The Dlt Header Type (HTYP) contains general information about the Dlt message.

[PRS_Dlt_00094] ⌈The Header Type (HTYP) shall contain the following information and be encoded the following way:

- Bit 0: UEH (Use Extended Header)
- Bit 1: MSBF (Most Significant Byte First)
- Bit 2: WEID (With ECU ID)
- Bit 3: WSID (With Session ID)
- Bit 4: WTMS (With Timestamp)
- Bit 5-7: VERS (Version Number)

⌋ (RS_LT_00002)

	Header Type (HTYP)							
Name	UEH	MSBF	WEID	WSID	WTMS	VERS		
Bit	0	1	2	3	4	5	6	7
Byte	**0**							

Figure 8 - Encoding of Header Type

[PRS_Dlt_00600] ⌈The Header Type (HTYP) shall be the first byte of any Dlt message. ⌋ (RS_LT_00002)

[PRS_Dlt_00601] ⌈The size of the Header Type (HTYP) shall be 1 byte. ⌋ (RS_LT_00002)

[PRS_Dlt_00602] ⌈If the UEH bit is set to '0', the extended header is not used. ⌋ (RS_LT_00002)

[PRS_Dlt_00603] ⌈If the UEH bit is set to '1', the extended header is used. ⌋ (RS_LT_00002)

Note:
If the verbose mode is needed, the UEH bit has to be set to 1.

[PRS_Dlt_00604] ⌈If the MSBF bit is set to '0', the payload data is encoded in little endian format. ⌋ (RS_LT_00002, RS_LT_00016)

[PRS_Dlt_00605] ⌈If the MSBF bit is set to '1', the payload data is encoded in big endian format. ⌋ (RS_LT_00002, RS_LT_00016)

[PRS_Dlt_00606] ⌈If the WEID bit is set to '0', the ECU ID (WEID) field is not contained in the Standard Header. ⌋ (RS_LT_00002, RS_LT_00022)

[PRS_Dlt_00607] ⌈If the WEID bit is set to ‘1’, the ECU ID (WEID) field is contained in the Standard Header. ⌋ (RS_LT_00002, RS_LT_00022)

[PRS_Dlt_00608] ⌈If the WSID bit is set to ‘0’, the Session ID (WSID) field is not contained in the Standard Header. ⌋ (RS_LT_00002)

[PRS_Dlt_00609] ⌈If the WSID bit is set to ‘1’, the Session ID (WSID) field is contained in the Standard Header. ⌋ (RS_LT_00002)

[PRS_Dlt_00610] ⌈If the WTMS bit is set to ‘0’, the Timestamp (WTMS) field is not contained in the Standard Header. ⌋ (RS_LT_00002)

[PRS_Dlt_00611] ⌈If the WTMS bit is set to ‘1’, the Timestamp (WTMS) field is contained in the Standard Header. ⌋ (RS_LT_00002)

[PRS_Dlt_00612] ⌈The VERS bits shall contain the Version number of the Dlt Data protocol in the Standard Header. ⌋ (RS_LT_00002)

Note:
Please be aware that the byte position of the defined fields within the Dlt Standard Header may shift due to the activation/deactivation of the optional fields (see above).

5.1.1.1.2 Message Counter

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

[PRS_Dlt_00319] ⌈The Message Counter is an unsigned 8-bit (0-255) integer. ⌋ (RS_LT_00018)

[PRS_Dlt_00613] ⌈After initialization of the Dlt module, the Message Counter (MCNT) shall be set to ‘0’. ⌋(RS_LT_00018)

[PRS_Dlt_00105] ⌈The Dlt module shall increment the Message Counter by one at every Log and Trace message received via the Dlt API.⌋ (RS_LT_00018)

[PRS_Dlt_00106] ⌈If the Message Counter reaches 255, the counter shall wrap around and start with the value ‘0’ at the next Log and Trace message to be transmitted.⌋ (RS_LT_00018)

5.1.1.1.3 Length

[PRS_Dlt_00320] ⌈The Length field LEN of the Dlt Standard Header shall be a 16-bit unsigned integer.⌋ (RS_LT_00002)

[PRS_Dlt_00614] ⌈The Length Field LEN of the Dlt Standard Header shall be set to the overall length of the Dlt message, which is the sum of the length of the Standard

Header itself, the length of the optional Dlt Extended Header, and the length of the optional Payload.⌋ (RS_LT_00002)

5.1.1.1.4 The optional ECU ID

The optional ECU ID is used to identify which ECU has sent a Dlt message. Therefore, it is highly recommended that the ECU ID is unique within the vehicle.

[PRS_Dlt_00321] ⌈The ECU ID shall be a 32-bit field interpreted as four 8-bit ASCII characters.⌋ (RS_LT_00022)

[PRS_Dlt_00308] ⌈If the ECU ID is shorter than four 8-bit ASCII characters, the remaining characters shall be filled with 0x00. ⌋ (RS_LT_00022)

[PRS_Dlt_00615] ⌈The ECU ID field shall only exist if the WEID bit of the Header Type is set to ‘1’).⌋ (RS_LT_00022)

5.1.1.1.5 Session ID

The optional Session ID is used to identify the source of a log or trace message within an ECU.

[PRS_Dlt_00322] ⌈The Session ID field shall be a 32-bit unsigned integer.⌋ (RS_LT_00020)

[PRS_Dlt_00616] ⌈The Session ID field shall only exist if the WSID bit of the Header Type is set to ‘1’. ⌋ (RS_LT_00020)

5.1.1.1.6 Timestamp

The optional Timestamp is used to add timing information a Dlt message has been generated.

[PRS_Dlt_00323] ⌈The Timestamp field (TMSP) shall be a 32-bit unsigned integer.⌋ (RS_LT_00017)

[PRS_Dlt_00112] ⌈The Timestamp field (TMSP) shall hold the timestamp from the moment an application sends the message to the Dlt module.⌋ (RS_LT_00017)

[PRS_Dlt_00309] ⌈The time resolution is in 0.1 milliseconds.⌋ (RS_LT_00017)

[PRS_Dlt_00628] ⌈The timer shall start with ‘00000000’ when the ECU which generates the LogMessage starts up. ⌋ (RS_LT_00017)

5.1.1.2 Extended Header

In case the UEH bit of the Standard Header is set to ‘1’, additional information is transmitted which are defined in the Dlt Extended Header format.

The Dlt Extended Header is directly attached after the Dlt Standard Header fields.

	Message Info	Number of Arguments	Application ID				Context ID			
Name	MSIN	NOAR	APID				CTID			
Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Figure 9 - Extended Header

[PRS_Dlt_00617] ⌈ The Dlt Extended Header format shall consist of the following fields in the following order:

- Byte 0: MSIN (Message Info)
- Byte 1: NOAR (Number of Arguments)
- Byte 2-5: APID (Application ID)
- Byte 6-9: CTID (Context ID)

⌋ (RS_LT_00002)

5.1.1.2.1 Message Info

[PRS_Dlt_00618] ⌈The Message Info field (MSIN) shall contain the following information in the following order:

- Bit 0: VERB (Verbose)
- Bit 1-3: MSTP (Message Type)
- Bit 4-7: MTIN (Message Type Info)

⌋ (RS_LT_00002)

	Message Info (MSIN)							
Name	VERB	MSTP			MTIN			
Bit offset	0	1	2	3	4	5	6	7
Byte	**0**							

Figure 10 - Encoding of the Message Info field

[PRS_Dlt_00119] ⌈If the VERB bit is set to ‘1’, the payload shall be transmitted in verbose mode.⌋ (RS_LT_00044)

[PRS_Dlt_00310] ⌈If the VERB bit is set to '0', the payload shall be transmitted in non-verbose mode.⌋ (RS_LT_00024)

[PRS_Dlt_00324] ⌈ The Message Type (MSTP) shall be a 3-bit unsigned integer. ⌋ (RS_LT_00002)

[PRS_Dlt_00120] ⌈The Message Type (MSTP) shall have one of the following values:

- 0x0: DLT_TYPE_LOG (Dlt Log Message)
- 0x1: DLT_TYPE_APP_TRACE (Dlt Trace Message)
- 0x2: DLT_TYPE_NW_TRACE (Dlt Network Message)
- 0x3: DLT_TYPE_CONTROL (Dlt Control Message)
- 0x4 – 0x7: *Reserved*

⌋ (RS_LT_00002)

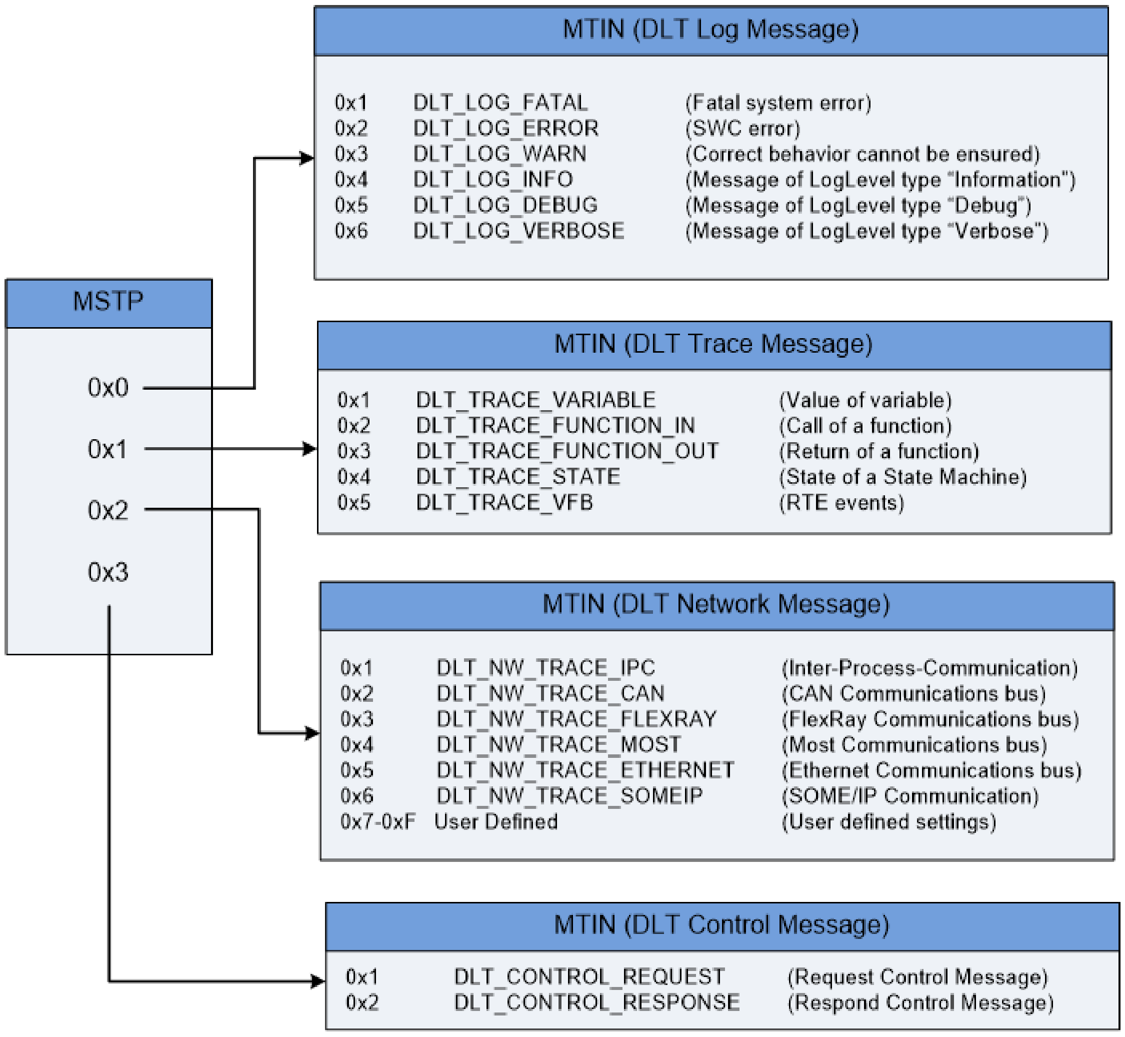

MSTP
0x0
0x1
0x2
0x3
MTIN (DLT Log Message)
0x1 DLT_LOG_FATAL (Fatal system error)
0x2 DLT_LOG_ERROR (SWC error)
0x3 DLT_LOG_WARN (Correct behavior cannot be ensured)
0x4 DLT_LOG_INFO (Message of LogLevel type "Information")
0x5 DLT_LOG_DEBUG (Message of LogLevel type "Debug")
0x6 DLT_LOG_VERBOSE (Message of LogLevel type "Verbose")
MTIN (DLT Trace Message)
0x1 DLT_TRACE_VARIABLE (Value of variable)
0x2 DLT_TRACE_FUNCTION_IN (Call of a function)
0x3 DLT_TRACE_FUNCTION_OUT (Return of a function)
0x4 DLT_TRACE_STATE (State of a State Machine)
0x5 DLT_TRACE_VFB (RTE events)
MTIN (DLT Network Message)
0x1 DLT_NW_TRACE_IPC (Inter-Process-Communication)
0x2 DLT_NW_TRACE_CAN (CAN Communications bus)
0x3 DLT_NW_TRACE_FLEXRAY (FlexRay Communications bus)
0x4 DLT_NW_TRACE_MOST (Most Communications bus)
0x5 DLT_NW_TRACE_ETHERNET (Ethernet Communications bus)
0x6 DLT_NW_TRACE_SOMEIP (SOME/IP Communication)
0x7-0xF User Defined (User defined settings)
MTIN (DLT Control Message)
0x1 DLT_CONTROL_REQUEST (Request Control Message)
0x2 DLT_CONTROL_RESPONSE (Respond Control Message)

Figure 11 - Dependency between the MSTP field and the MTIN field

[PRS_Dlt_00325] ⌈The Message Type Info field (MTIN) shall be a 4-bit unsigned integer. ⌋ (RS_LT_00002)

[PRS_Dlt_00619] ⌈If the MSTP field is set to 0x0 (i.e. Dlt Log Message), the Message Type Info field (MTIN) shall have one of the following values with the following meaning:

- 0x1: DLT_LOG_FATAL (Fatal system error)
- 0x2: DLT_LOG_DLT_ERROR (Application error)
- 0x3: DLT_LOG_WARN (Correct behavior cannot be ensured)
- 0x4: DLT_LOGINFO (Message of LogLevel type “Information”)
- 0x5: DLT_LOG_DEBUG (Message of LogLevel type “Debug”)
- 0x6: DLT_LOG_VERBOSE (Message of LogLevel type “Verbose”)
- 0x7 – 0xF: *Reserved* ⌋ (RS_LT_00002)

[PRS_Dlt_00620] ⌈If the MSTP field is set to 0x1 (i.e. Dlt Trace Message), the Message Type Info field (MTIN) shall have one of the following values with the following meaning:

- 0x1: DLT_TRACE_VARIABLE (Value of variable)
- 0x2: DLT_TRACE_FUNCTION_IN (Call of a function)
- 0x3: DLT_TRACE_FUNCTION_OUT (Return of a function)
- 0x4: DLT_TRACE_STATE (State of a State Machine)
- 0x5: DLT_TRACE_VFB (RTE events)
- 0x6 – 0xF: *Reserved* ⌋ (RS_LT_00002)

[PRS_Dlt_00621] ⌈If the MSTP field is set to 0x2 (i.e. Dlt Network Message), the Message Type Info field (MTIN) shall have one of the following values with the following meaning:

- 0x1: DLT_NW_TRACE_IPC (Inter-Process-Communication)
- 0x2: DLT_NW_TRACE_CAN (CAN Communications bus)
- 0x3: DLT_NW_TRACE_FLEXRAY (FlexRay Communications bus)
- 0x4: DLT_NW_TRACE_MOST (Most Communications bus)
- 0x5: DLT_NW_TRACE_ETHERNET (Ethernet Communications bus)
- 0x6: DLT_NW_TRACE_SOMEIP (Inter-SOME/IP Communication)
- 0x7-0xF: *User Defined* (User defined settings) ⌋ (RS_LT_00002)

[PRS_Dlt_00622] ⌈The If the MSTP field is set to 0x3 (i.e. Dlt Control Message), the Message Type Info field (MTIN) shall have one of the following values with the following meaning:

- 0x1: DLT_CONTROL_REQUEST (Request Control Message)
- 0x2: DLT_CONTROL_RESPONSE (Respond Control Message)
- 0x3-0xF: *Reserved* ⌋ (RS_LT_00002)

5.1.1.2.2 Number of Arguments

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

[PRS_Dlt_00326] ⌈ The Number of Arguments field (NOAR) shall be an 8-bit unsigned integer. ⌋ (RS_LT_00002)

[PRS_Dlt_00126] ⌈ If the VERB bit is set to ‘1’ (i.e. Verbose Mode is used), the Number of Arguments field (NOAR) shall contain the number of provided arguments within the payload. ⌋ (RS_LT_00002, RS_LT_00024)

[PRS_Dlt_00623] ⌈ If the VERB bit is set to ‘0’ (i.e. Verbose Mode is not used), the Number of Arguments field (NOAR) shall be set to 0x00. ⌋ (RS_LT_00002, RS_LT_00023)

5.1.1.2.3 Application ID

The Application ID is an abbreviation of the application, which generates the Dlt message.

[PRS_Dlt_00127] ⌈The Application ID field (APID) shall be a 32-bit field interpreted as four 8-bit ASCII characters.⌋ (RS_LT_00021)

[PRS_Dlt_00312] ⌈If the Application ID is shorter than four 8-bit ASCII characters, the remaining characters shall be filled with 0x00. ⌋ (RS_LT_00021)

5.1.1.2.4 Context ID

The Context ID is a user defined ID to (logically) group Dlt messages generated by an application.

[PRS_Dlt_00128] ⌈The Context ID field (CTID) shall be a 32-bit field interpreted as four 8-bit ASCII characters. ⌋ (RS_LT_00021)

[PRS_Dlt_00313] ⌈If the Context ID is shorter than four 8-bit ASCII characters, the remaining characters shall be filled with 0x00. ⌋ (RS_LT_00021)

5.1.2 Body/Payload format

The Dlt Payload follows the Dlt Header or the Dlt Extended Header if used. The Dlt Payload contains the parameters that are logged or traced, or it contains control information.

[PRS_Dlt_00314] ⌈If the UEH (Use Extended Header) bit is set to ’1’, the payload shall adjoin the Dlt Extended Header. ⌋ (RS_LT_00013, RS_LT_00023)
[PRS_Dlt_00315] ⌈If the UEH (Use Extended Header) bit is set to ’0’, the payload shall adjoin the Dlt Standard Header. ⌋ (RS_LT_00013, RS_LT_00023)

5.1.2.1 Non-Verbose Mode

To be able to transmit parameter values only - without the need of any meta information about them -, additional properties like parameter names or types -, the Non-Verbose Mode can be used.

To allow the correct disassembly of the contained parameter values within a received Dlt message, a dedicated Message ID is added to the payload.

A separate, external file contains the description of the payload layout according to the corresponding Message ID.

Standard Header	Payload	
	Message ID	Non-Static Data

Figure 12 – Non-Verbose Mode message

[PRS_Dlt_00624] ⌈The Message ID shall be a 32-bit unsigned integer. ⌋ (RS_LT_00027)

[PRS_Dlt_00352] ⌈The Message ID shall be assigned unique for a single combination of static data. ⌋ (RS_LT_00027)

[PRS_Dlt_00353] ⌈With the combination of a Message ID and an external description, following information shall be recoverable: the Type Info:

- Type Length
- Data Type
- String Coding
- Variable Info
- Fixed Point

⌋ (RS_LT_00023, RS_LT_00026)

Note: If verbose mode is used instead (see chapter 7.2.5), these parameters are contained directly within the Dlt message payload.

[PRS_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)

⌋ (RS_LT_00023, RS_LT_00026)

5.1.2.1.1 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 1 - 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.

[PRS_Dlt_00378] ⌈The non-static data shall be transmitted consistently packed and byte aligned. ⌋ (RS_LT_00014, RS_LT_00023)

5.1.2.1.2 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 ARXML System Description shall be used. Please note that the description is currently only supported in the AUTOSAR Classic Platform.

The software supplier of an application or of the middleware shall provide this description file. Please note that DLT can have several sources of log or trace messages (several applications or diagnostic modules) and therefore the ECU in the System Description may aggregate several **DltLogChannel**s. Each **DltLogChannel** of the ECU points to **DltMessage**s that represent the NonVerbose messages of this ECU.

Some information from the Extended Header is described by the **DltLogChannel** (applicationId and contextId). The Message Info and the Number of Arguments can be derived from the **DltMessage** itself.

As seen from the user, a log or trace message consists of several arguments. Each argument can be either:

- static text or
- non static variable.

Please note that only non-static variables are transmitted. To reassemble the whole message with all arguments, the static text arguments shall be described as well. In the ARXML System Description the **DltMessage** consists of an ordered list of **DltArgument**s.

If a **DltArgument** is a static text, then the **DltArgument** shall only contain a ShortName and a Description.
If a **DltArgument** represents a non-static variable then **DltArgument.***length*, the unit and the baseType shall be defined.

[PRS_Dlt_00396] ⌈One log or trace message shall be represented by one **DltMessage** element. ⌋ (RS_LT_00026)

[PRS_Dlt_00397] ⌈The Message ID shall be set in **DltMessage**.*messageId*⌋ (RS_LT_00026, RS_LT_00027)

[PRS_Dlt_00398] ⌈The DLT message is described with the following information:

- Message Type (MSTP) – shall be derived from **DltMessage**.*messageTypeInfo*
- Message Info (MSIN) – **DltMessage**.*messageTypeInfo*
- Application ID (APID) – **DltLogChannel**.*applicationId*
- Context ID (CTID) – **DltLogChannel**.*contextId*
- Source file – **DltMessage**.*messageSourceFile*
- line number – **DltMessage**.*messageLineNumber* ⌋

(RS_LT_00026, RS_LT_00027)

[PRS_Dlt_00399] ⌈The user data of the log or trace message shall be represented by **DltArgument**s that are ordered in the **DltMessage**. ⌋ (RS_LT_00026)

[PRS_Dlt_00400] ⌈If the argument contains only static text, this text shall be placed in the DESC element of the **DltArgument**. In this case the **DltArgument**.*length* shall be set to zero.⌋ (RS_LT_00025)

[PRS_Dlt_00401] ⌈If the argument contains non-static data, the data transported in the message is described by the **DltArgument**.*length* and by the aggregated **networkRepresentation** that defines the unit and the baseType.⌋ (RS_LT_00026)

The following example shows the description of a sample Dlt message in ARXML:

```
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://autosar.org/schema/r4.0
                                   autosar 00048.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Dlt Example</SHORT-NAME>
      <ELEMENTS>
        <DLT-MESSAGE-COLLECTION-SET>
          <SHORT-NAME>
            DltMessageCollectionSetExample
          </SHORT-NAME>
          <DLT-MESSAGES>
            <DLT-MESSAGE>
              <SHORT-NAME>DltMessageWithID 1</SHORT-NAME>
              <DLT-ARGUMENTS>
                <DLT-ARGUMENT>
                  <SHORT-NAME>
                    Temperature measurement
                  </SHORT-NAME>
                  <DESC>
                    <L-2 L="EN">
                      Temperature measurement
                    </L-2>
                  </DESC>
                  <LENGTH>0</LENGTH>
                </DLT-ARGUMENT>
                <DLT-ARGUMENT>
                  <SHORT-NAME>measurement point</SHORT-NAME>
                  <DESC>
                    <L-2 L="EN">
                      Temperature measurement
                    </L-2>
                  </DESC>
                  <LENGTH>8</LENGTH>
                  <NETWORK-REPRESENTATION>
                    <SW-DATA-DEF-PROPS-VARIANTS>
                      <SW-DATA-DEF-PROPS-CONDITIONAL>
                        <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                          /AUTOSAR Platform/BaseTypes/uint8
                        </BASE-TYPE-REF>
                        <UNIT-REF DEST="UNIT">
                          /AUTOSAR/AISpecification/Units/NoUnit
                        </UNIT-REF>
                      </SW-DATA-DEF-PROPS-CONDITIONAL>
                    </SW-DATA-DEF-PROPS-VARIANTS>
                  </NETWORK-REPRESENTATION>
                </DLT-ARGUMENT>
                <DLT-ARGUMENT>
```

```
                  <SHORT-NAME>reading</SHORT-NAME>
                  <DESC><L-2 L="EN">reading</L-2></DESC>
                  <LENGTH>8</LENGTH>
                  <NETWORK-REPRESENTATION>
                    <SW-DATA-DEF-PROPS-VARIANTS>
                      <SW-DATA-DEF-PROPS-CONDITIONAL>
                        <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                          /AUTOSAR Platform/BaseTypes/float32
                        </BASE-TYPE-REF>
                        <UNIT-REF DEST="UNIT">
                          /AUTOSAR/AISpecification/Units/Kelvin
                        </UNIT-REF>
                      </SW-DATA-DEF-PROPS-CONDITIONAL>
                    </SW-DATA-DEF-PROPS-VARIANTS>
                  </NETWORK-REPRESENTATION>
                </DLT-ARGUMENT>
              </DLT-ARGUMENTS>
              <MESSAGE-ID>1</MESSAGE-ID>
              <MESSAGE-LINE-NUMBER>72</MESSAGE-LINE-NUMBER>
              <MESSAGE-SOURCE-FILE>
                demo.c
              </MESSAGE-SOURCE-FILE>
              <MESSAGE-TYPE-INFO>
                DLT LOG DEBUG
              </MESSAGE-TYPE-INFO>
            </DLT-MESSAGE>
          </DLT-MESSAGES>
        </DLT-MESSAGE-COLLECTION-SET>
        <ECU-INSTANCE>
          <SHORT-NAME>TestEcu</SHORT-NAME>
          <DLT-CONFIG>
            <DLT-ECU-ID>TestEcu</DLT-ECU-ID>
            <DLT-LOG-CHANNELS>
              <DLT-LOG-CHANNEL>
                <SHORT-NAME>LogChannelCan</SHORT-NAME>
                <APPLICATION-ID>APPI</APPLICATION-ID>
                <CONTEXT-ID>CONI</CONTEXT-ID>
                <DLT-MESSAGE-REFS>
                  <DLT-MESSAGE-REF DEST="DLT-MESSAGE">
/Dlt Example/DltMessageCollectionSet/DltMessageWithID 1
                  </DLT-MESSAGE-REF>
                </DLT-MESSAGE-REFS>
              </DLT-LOG-CHANNEL>
            </DLT-LOG-CHANNELS>
          </DLT-CONFIG>
        </ECU-INSTANCE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

5.1.2.2 Verbose Mode

Dlt messages which are sent in Verbose Mode contain a complete description of the parameters next to the parameter values itself.
This means that on the one hand no external file is needed for disassembly; On the other hand, a higher amount of data is sent on the bus.

The Verbose Mode can be used on ECUs 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 external information.

5.1.2.2.1 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 itself. 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 within the Number of arguments field (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.

[PRS_Dlt_00459] ⌈The Dlt message in Verbose Mode shall consist of
- Standard Header
- Extended Header
- n Arguments, each consisting of a tuple of Type Info and Data Payload

⌋ (RS_LT_00023, RS_LT_00044)

Standard Header	Extended Header	**Payload**			
		Argument 1		Argument n	
		Type Info	Data Payload	**Type Info**	Data Payload

Figure 13 –Verbose Mode message

[PRS_Dlt_00409] ⌈The arguments and all inherited data shall be transmitted consistently packed. ⌋ (RS_LT_00023, RS_LT_00013)

5.1.2.2.2 Data Payload

The Data Payload contains the value of the variable (i.e. the debug information of an application or middleware), which is going to be transmitted on the communications bus. In addition to the variable value itself, it is needed to provide information like size and type of the variable. This information is contained in the Type Info field

5.1.2.2.3 Type Info

The Type Info Filed contains meta data about the Data Payload

[PRS_Dlt_00135] ⌈The Type Info is a 32 bit field and has to be part of the Payload segment if a Dlt log or trace message shall be sent in Verbose Mode ⌋ (RS_LT_00002, RS_LT_00044)

[PRS_Dlt_00625] ⌈The Type Info is a 32 bit field shall be encoded the following way:

• **Bit 0 - 3**	**Type Length (TYLE)**
• Bit 4	Type Bool (BOOL)
• Bit 5	Type Signed (SINT)
• Bit 6	Type Unsigned (UINT)
• Bit 7	Type Float (FLOA)
• Bit 8	Type Array (ARAY)
• Bit 9	Type String (STRG)
• Bit 10	Type Raw (RAWD)
• Bit 11	Variable Info (VARI)
• Bit 12	Fixed Point (FIXP)
• Bit 13	Trace Info (TRAI)
• Bit 14	Type Struct (STRU)
• **Bit 15 – 17**	**String Coding (SCOD)**
• *Bit 18 – 31*	*reserved for future use*

⌋(RS_LT_00044)

	Type Info (4 bytes)													
Name	Type Length (TYLE)	Type Bool (Bool)	Type Signed (SINT)	Type Unsigned (UINT)	Type Float (FLOA)	Type Array (ARAY)	Type String (STRG)	Type Raw (RAWD)	Variable Info (VARI)	Fixed Point (FIXP)	Trace Info (TRAI)	Type Struct (STRU)	String Coding (SCOD)	*reserved*
Bit	0-3	4	5	6	7	8	9	10	11	12	13	14	15-17	18-31

Figure 14– Encoding of the Type Info bit field

[PRS_Dlt_00626] ⌈The bits 0-3 (i.e. Type Length) of the Type Info field define the length of the adjoined Data Payload. The Type Length (TYLE) bit-field is encoded the following way:

• 0x00:	not defined
• 0x01:	8 bit
• 0x02:	16 bit
• 0x03:	32 bit
• 0x04:	64 bit
• 0x05:	128 bit
• *0x06 – 0x0F:*	*reserved*

⌋(RS_LT_00044)

[PRS_Dlt_00627] ⌈The bits 15-17 (i.e. String Coding) of the Type Info field define the String type. It is encoded the following way:

• 0x00:	ASCII
• 0x01:	UTF-8
• *0x02-0x07:*	*reserved*

⌋(RS_LT_00044, RS_LT_00025)

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 5-1 Simplified Assembly of Type Info

The entries of Type Info are specified in the following section in detail.

Details regarding the Data Types of the Type Info field are described in the following chapter.

5.1.3 Data Types

5.1.3.1 *Bits* Type Length (TYLE)

Type Length specifies the length of the standard data type.

[PRS_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) ⌋ (RS_LT_00044)

5.1.3.2 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.

[PRS_Dlt_00410] ⌈The coding of all text in Variable Info (VARI) shall be in 8-bit ASCII format. ⌋ (RS_LT_00025)

[PRS_Dlt_00411] ⌈The strings in VARI shall be null terminated. ⌋ (RS_LT_00044, RS_LT_00025)

5.1.3.3 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.

[PRS_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 ⌋ (RS_LT_00044)

[PRS_Dlt_00169] ⌈The bit Fixed Point (FIXP) shall only be set in combination with Type Signed (SINT) or Type Unsigned (UINT). ⌋ (RS_LT_00044)

5.1.3.4 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.

[PRS_Dlt_00182] ⌈String Coding is a bit field of 3 bit. ⌋ (RS_LT_00044, RS_LT_00025)

[PRS_Dlt_00366] ⌈Following values shall be used for String Coding (SCOD):
0x00 = ASCII
0x01 = UTF-8
0x02 - 0x07 reserved for future use ⌋ (RS_LT_00044, RS_LT_00025)

[PRS_Dlt_00183] ⌈String Coding shall be set if Type String (STRG) is set. ⌋ (RS_LT_00044, RS_LT_00025)

[PRS_Dlt_00367] ⌈String Coding shall be set if Trace Info (TRAI) is set. ⌋ (RS_LT_00044, RS_LT_00025)

5.1.3.5 Type Bool (BOOL)

[PRS_Dlt_00422] ⌈If the BOOL bit is set, the Data Payload shall consist of at least one 8-bit unsigned integer parameter. ⌋ (RS_LT_00044)

[PRS_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. ⌋ (RS_LT_00044)

[PRS_Dlt_00139] ⌈Type Length (TYLE) shall be 1. ⌋ (RS_LT_00044)

[PRS_Dlt_00355] ⌈If Variable Info (VARI) is set, the Length of Name, the Name and the Unit fields shall be added. ⌋ (RS_LT_00044)

[PRS_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 5-2 Data Payload of Type Bool (BOOL)

⌋ (RS_LT_00044)

5.1.3.6 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.

[PRS_Dlt_00385] ⌈If the SINT bit is set, the Data Payload consists of at least one signed integer Data field. ⌋ (RS_LT_00044)

[PRS_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

Length in bit	Name	Description
8/16/32 / 64 / 128	**Data**	Length depends on TYLE

Table 5-3 Data Payload of Type Signed (SINT) and Type Unsigned (UINT)

⌋ (RS_LT_00044)

[PRS_Dlt_00356] ⌈Type Length (TYLE) shall be set to 1, 2, 3, 4 or 5. ⌋ (RS_LT_00044)

[PRS_Dlt_00357] ⌈If Variable Info (VARI) is set, the "Length of Name", "Length of Unit", the "Name" and the "Unit" fields shall be added. ⌋ (RS_LT_00044)

[PRS_Dlt_00412] ⌈If FIXP is set, the Quantization and Offset fields shall be added. ⌋ (RS_LT_00044)

[PRS_Dlt_00388] ⌈The Quantization field shall be a 32-bit float field. ⌋ (RS_LT_00044)

[PRS_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. ⌋ (RS_LT_00044)

[PRS_Dlt_00358] ⌈The length of Data shall depend on Type Length (TYLE). ⌋ (RS_LT_00044)

[PRS_Dlt_00370] ⌈The Data Payload of Type Signed (SIGN) and of Type Unsigned (UINT) shall be assembled as shown in Table 5-3. ⌋ (RS_LT_00044)

5.1.3.7 Type Float (FLOA)

[PRS_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

Length in bit		**Name**	**Description**
	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 5-4 Data Payload of Type Float (FLOA)

⌋ (RS_LT_00044)

[PRS_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 5-5 Definition of Type Length according to IEEE 754r

⌋ (RS_LT_00044)

[PRS_Dlt_00362] ⌈If Variable Info (VARI) is set, the “Length of Name”, “Length of Unit”, the “Name” and the “Unit” fields shall be added. ⌋ (RS_LT_00044)

[PRS_Dlt_00363] ⌈The length of Data shall depend on Type Length (TYLE). ⌋ (RS_LT_00044)

[PRS_Dlt_00371] ⌈The argument of Type Float (FLOA) shall be assembled as shown in Table 5-4. ⌋ (RS_LT_00044)

5.1.3.8 Type String (STRG)

[PRS_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.⌋ (RS_LT_00025)

[PRS_Dlt_00155] ⌈String Coding (SCOD) shall be specified. ⌋ (RS_LT_00025)

[PRS_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. ⌋ (RS_LT_00025)

[PRS_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. ⌋ (RS_LT_00025)

[PRS_Dlt_00157] ⌈If Variable Info (VARI) is set, the "Length of Name" and the "Name" fields shall be added. ⌋ (RS_LT_00025)

[PRS_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 5-6 Data Payload of Type String (STRG)

⌋ (RS_LT_00025)

5.1.3.9 Type Array (ARAY)

[PRS_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. ⌋ (RS_LT_00044)

[PRS_Dlt_00148] ⌈At the beginning of the Data Payload a 16-bit unsigned integer shall specify the number of dimensions of the array. ⌋ (RS_LT_00044)

[PRS_Dlt_00149] [If Variable Info (VARI) is set, the name of the array shall be described.⌋ (RS_LT_00044)

[PRS_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. ⌋ (RS_LT_00044)

[PRS_Dlt_00152] ⌈If Variable Info (VARI) is set, the "Length of Name", "Length of Unit", the "Name" and the "Unit" fields shall be added. ⌋ (RS_LT_00044)

[PRS_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. ⌋ (RS_LT_00044)

[PRS_Dlt_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.	

Table 5-7 Data Payload of Type Array (ARAY)

⌋ (RS_LT_00044)

5.1.3.10 Type Struct (STRU)

If this bit is set, structured data are transmitted.

[PRS_Dlt_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. ⌋ (RS_LT_00044)

[PRS_Dlt_00176] ⌈If Variable Info (VARI) is set, the “Length of Name” and the “Name” fields shall be added. ⌋ (RS_LT_00044)

[PRS_Dlt_00177] ⌈The list of entries contains one or more standard arguments with Type Info and Data Payload. All standard argument types are allowed. ⌋ (RS_LT_00044)

[PRS_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 5-8 Data Payload of Type Struct (STRU)

⌋ (RS_LT_00044)

5.1.3.11 Type Raw (RAWD)

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

[PRS_Dlt_00364] ⌈If Variable Info (VARI) is set, the coding of the name shall be in 8-bit ASCII format. ⌋ (RS_LT_00044)

[PRS_Dlt_00160] ⌈At the beginning of the Data Payload a 16-bit unsigned integer shall specify the length of the raw data in byte. ⌋ (RS_LT_00044)

[PRS_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 cannot be done. Some tools can show this data by a user defined data type. ⌋ (RS_LT_00044)

[PRS_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 5-9 Data Payload of Type Raw (RAWD)

⌋ (RS_LT_00044)

5.1.3.12 Type Trace Info (TRAI)

Trace info is a separate argument in the Dlt message.

[PRS_Dlt_00170] ⌈If the bit Trace Info (TRAI) is set, the trace information (like module name / function) shall be transmitted in the argument. ⌋ (RS_LT_00044)

[PRS_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. ⌋ (RS_LT_00044)

[PRS_Dlt_00173] ⌈Null terminated trace data string shall follow. ⌋ (RS_LT_00044)

[PRS_Dlt_00171] ⌈String Coding (SCOD) shall specify the coding of the trace data string. ⌋ (RS_LT_00044)

[PRS_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 5-10 Data Payload of Trace Info (TRAI)

⌋ (RS_LT_00044)

5.1.3.13 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 5-11 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								
											o			x					
											o								

Table 5-12 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 5-13 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 application.

5.1.3.14 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).

[PRS_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. ⌋ (RS_LT_00026)

[PRS_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. ⌋ (RS_LT_00026)

5.2 Message types

5.2.1 Data Messages

Dlt Data Messages are assembled as described in chapter 5.1 “Message format”.

5.2.2 Control Messages

Dlt Control Messages are mainly used to modify the behavior of the Dlt module at runtime. They allow things like changing the communications bus to send Dlt data messages, modifying the filter leve, configuration can be triggered to be stored nonvolatile.

5.3 Services / Commands

The following chapters describe the defined Dlt Commands, including an unique ID (Service ID), the format, and the required parameters.

[PRS_Dlt_00635] ⌈The following Dlt Commands using the following Services IDs shall be supported:

Service ID	Dlt Command Name	Description
0x01	SetLogLevel	Set the Log Level
0x02	SetTraceStatus	Enable/Disable Trace Messages
0x03	GetLogInfo	Returns the LogLevel for applications
0x04	GetDefaultLogLevel	Returns the LogLevel for wildcards
0x05	StoreConfiguration	Stores the current configuration non volatile
0x06	RestoreToFactoryDefault	Sets the configuration back to default
0x0A	SetMessageFiltering	Enable/Disable message filtering
0x11	SetDefaultLogLevel	Sets the LogLevel for wildcards
0x12	SetDefaultTraceStatus	Enable/Disable TraceMessages for wildcards
0x13	GetSoftwareVersion	Get the ECU software version
0x15	GetDefaultTraceStatus	Get the current TraceLevel for wildcards
0x17	GetLogChannelNames	Returns the LogChannel's name
0x1F	GetTraceStatus	Returns the current TraceStatus
0x20	SetLogChannelAssignment	Adds/ Removes the given LogChannel as output path
0x21	SetLogChannelThreshold	Sets the filter threshold for the given LogChannel
0x22	GetLogChannelThreshold	Returns the current LogLevel for a given LogChannel
0x23	BufferOverflowNotification	Report that a buffer overflow occurred
0x24	SyncTimeStamp	Reports synchronized absolute time

⌋ (RS_LT_00032)

Note:

It is recommended that the defined Dlt Commands can be triggered by the reception of the corresponding Dlt Control Message, and/or via separate C APIs.

[PRS_Dlt_00187] ⌈Control messages are normal Dlt messages with a Standard Header, an Extended Header, and payload. The payload consists of the Service ID, transmitted as 32-bit unsigned integer and the contained parameters.

5.3.1 Set Log Level

[PRS_Dlt_00194] ⌈

Service name:	SetLogLevel		
Service_ID [hex]	`0x00000001`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	**applicationId**	`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	**contextId**	`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	**newLogLevel**	`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	***reserved***	`4*uint8`	Reserved – These 4 bytes shall be ignored (i.e.: "don't care") This field shall be filled with zeros.
Response Parameter			
Number	***Name***	**Type**	**Description**
1	**status**	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Set the pass through range for log messages for a given combination of Application ID/ Context ID.		

⌋ (RS_LT_00032)

[PRS_Dlt_00195] ⌈Action to process:
Update the LogLevel setting within the Dlt module and inform all registered Applications with the Application ID which has been provided by the Dlt_SetLogLevel service.⌋ (RS_LT_00032)

5.3.2 Set Trace Status

[PRS_Dlt_00196] ⌈

Service name:	SetTraceStatus		
Service ID [hex] ***:***	0x00000002		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	**applicationId**	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	**contextId**	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	**newTraceStatus**	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	***reserved***	4 bytes	Reserved - These 4 bytes shall be ignored (i.e.: “don’t care”). This field shall be filled with zeros.
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 tuple of Application ID / Context ID.		

⌋ (RS_LT_00032)

5.3.3 Get Log Info

[PRS_Dlt_00197] ⌈

Service name:	GetLogInfo		
Service ID [hex]	`0x00000003`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
1	**options**	`uint8`	1 - reserved 2 - reserved 3 - reserved 4 - reserved 5 - unused – Information about unregistered Application IDs and Context IDs cannot be requested **6** - Information about registered Application IDs and Context IDs with log level and with trace status information **7** - 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
2	**applicationId**	`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	**contextId**	`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	***reserved***	`4*uint8`	Reserved – These 4 bytes shall be ignored (i.e.: “don’t care”) This field shall be filled with zeros.
Response Parameter			
Number	***Name***	**`Type`**	**Description**

1	**status**	`uint8`	**1** - NOT_SUPPORTED **2** - DLT_ERROR 3 - reserved 4 - reserved **5** - Information about unregistered Application IDs and Context IDs **6** - Information about registered Application IDs and Context IDs with log level and with trace status information **7** - 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 too large. 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	***applicationIds***	`LogInfoType`	NULL if status == 1 or 2
			
			For status 6 or 7 (7 prefixed with '):
			appIdCount (uint16)
			appIdInfo[] (struct[])
			  appID (uint8[4])
			  contextIdCount (uint16)
			  contextIdInfoList[] (struct[])
			    contextId (uint8[4])
			    logLevel (enum 0x00 .. 0x06)
			    traceStatus (uint8)
			' lenContextDescription (uint16)
			' contextDesc[] (uint8[])
			' appDescLen (uint16)
			' appDesc[] (uint8[])

3	***reserved***	`4*uint8`	Reserved – These 4 bytes shall be ignored (i.e.: "don't care"). This field shall be filled with zeros.
Description:	Called to request information about registered Applications and their Contexts including the corresponding IDs, log levels, trace statuses and descriptions if requested. Also used to report added or deleted registrations of Application IDs and Context IDs. If the command GetLogInfo has been requested with a "reserved" options value, NOT_SUPPORTED shall be returned.		

⌋ (RS_LT_00032, RS_LT_00033)

5.3.4 Get Default Log Level

[PRS_Dlt_00198] ⌈

Service name:	GetDefaultLogLevel		
Service ID [hex]:	`0x00000004`		
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	***logLevel***	`uint8`	Actual log level
Description:	Returns the actual default log level.		

⌋ (RS_LT_00032)

5.3.5 Store Configuration

[PRS_Dlt_00199] ⌈

Service name:	StoreConfiguration		
Service ID [hex]:	`0x00000005`		
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 Dlt configuration nonvolatile. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032, RS_LT_00039)

5.3.6 Reset to Factory Default

[PRS_Dlt_00200] ⌈

Service name:	ResetToFactoryDefault		
Service ID [hex]:	`0x00000006`		
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 Dlt configuration back to factory defaults. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

5.3.7 SetMessageFiltering

[PRS_Dlt_00205] ⌈

Service name:	SetMessageFiltering		
Service ID [hex]:	`0x0000000A`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	**newstatus**	`Uint8`	0 – OFF 1 – ON
Response Parameter			
Number	***Name***	**Type**	**Description**
2	***status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Called to switch on/off the message filtering by the Dlt module. If not supported, NOT_SUPPORTED shall be the response		

⌋ (RS_LT_00040)

5.3.8 Set Default LogLevel

[PRS_Dlt_00380] ⌈

Service name:	SetDefaultLogLevel		
Service_ID [hex]	`0x00000011`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
1	***newLogLevel***	`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 • if set to -1 all messages pass the filter
4	***Reserved***	`4*uint8`	Reserved – These 4 bytes shall be ignored (i.e.: “don’t care”). This field shall be filled with zeros.
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. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

[PRS_Dlt_00381] ⌈Action to process:
Update the LogLevel filter for all wildcard entries according to the provided newLogLevel.⌋ (RS_LT_00032)

5.3.9 Set Default Trace Status

[PRS_Dlt_00383] ⌈

Service name:	SetDefaultTraceStatus		
Service_ID [hex]	`0x00000012`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
1	***newTraceStatus***	`sint8`	the new trace status to set • can be 1 – for On and 0 – for Off
2	***reserved***	`4 bytes`	These 4 bytes shall be ignored (i.e.: “don’t care”) This field shall be filled with zeros.
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. If not supported, NOT_SUPPORTED shall be the response.

⌋ (RS_LT_00032)

5.3.10 Get ECU Software Version

[PRS_Dlt_00393] ⌈

Service name:	GetSoftwareVersion		
Service_ID [hex]	`0x00000013`		
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	***Length***	`uint32`	Length of the string swVersion
3	***swVersion***	`char[]`	String containing the ECU software version
Description:	Getting the ECU's software version		

⌋ (RS_LT_00032)

5.3.11 Get Default Trace Status

[PRS_Dlt_00494] ⌈

Service name:	GetDefaultTraceStatus		
Service ID [hex]:	`0x00000015`		
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	***traceStatus***	`uint8`	Actual Trace Status 0 - off, 1 - on
Description:	Returns the actual default trace status. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

5.3.12 Get LogChannel Names

[PRS_Dlt_00502] ⌈

Service name:	GetLogChannelNames		
Service ID [hex]:	`0x00000017`		
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	***countIf***	`uin8`	Count of transmitted interface (i.e. LogChannel) names.
3	***logChannelNames***	`4*uin8[]`	List of Log Channel names. Array on each 4 byte
Description:	Called to get all available communication interfaces. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

5.3.13 Get Trace Status

[PRS_Dlt_00638] ⌈

Service name:	GetTraceStatus		
Service ID [hex]:	`0x0000001F`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
1	***applicationId***	`4*uint8`	Addressed Application ID
2	***contextId***	`4*uint8`	Addressed Context ID
Response Parameter			
Number	***Name***	**`Type`**	**Description**
1	***status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***traceStatus***	`uint8`	Actual Trace Status 0 - off, 1 - on
Description:	Returns the actual trace status for the addressed tuple of ApplicationID/ContextID. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

5.3.14 Set LogChannel Assignment

[PRS_Dlt_00637] ⌈

Service name:	SetLogChannelAssignment		
Service ID [hex]:	`0x00000020`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
1	***applicationId***	`4*uint8`	Addressed Application ID
2	***contextId***	`4*uint8`	Addressed Context ID
3	***logChannelName***	`4*uint8`	Name of the addressed LogChannel
4	***addRemoveOp***	`uint8`	**0:** Remove the addressed tuple of ApplicationID/Context ID from the addressed LogChannel 1: Add the addressed tuple of ApplicationID/ContextID to the addressed LogChannel
Response Parameter			
Number	***Name***	**`Type`**	**Description**
1	***status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
Description:	Adds or removes the addressed tuple of ApplicationID/ContextID from the addressed LogChannel. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

5.3.15 Set LogChannel Threshold

[PRS_Dlt_00639] ⌈

Service name:	SetLogChannelThreshold		
Service ID [hex]:	`0x00000021`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
1	***logChannelName***	`4*uint8`	Name of the addressed LogChannel
2	***logLevelThreshold***	`uint8`	**0** - DLT_LOG_OFF **1** - DLT_LOG_FATAL **2** - DLT_LOG_ERROR **3** - DLT_LOG_WARN **4** - DLT_LOG_INFO **5** - DLT_LOG_DEBUG **6** - DLT_LOG_VERBOSE
3	***traceStatus***	`uint8`	0: Trace Messages blocked 1: Trace Messages can pass
Response Parameter			
Number	***Name***	**`Type`**	**Description**
1	***Status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***traceStatus***	`uint8`	Actual Trace Status 0 - off, 1 - on
Description:	Sets the LogLevel and the TraceStatus for the addressed LogChannel. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

5.3.16 Get LogChannel Threshold

[PRS_Dlt_00640] ⌈

Service name:	GetLogChannelThreshold		
Service ID [hex]:	`0x00000022`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**`Type`**	**Description**
1	***logChannelName***	`4*uint8`	Name of the addressed LogChannel
Response Parameter			
Number	***Name***	**`Type`**	**Description**

1	***status***	`uint8`	0 == OK 1 == NOT_SUPPORTED 2 == ERROR
2	***logLevelThreshold***	`uint8`	0 == DLT_LOG_OFF 1 == DLT_LOG_FATAL 2 == DLT_LOG_ERROR 3 == DLT_LOG_WARN 4 == DLT_LOG_INFO 5 == DLT_LOG_DEBUG 6 == DLT_LOG_VERBOSE
3	***traceStatus***	`uint8`	0 == Trace Messages are blocked 1 == Trace Messages can pass
Description:	Returns the LogLevel and the TraceStatus for the addressed LogChannel. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00032)

5.3.17 BufferOverflowNotification

[PRS_Dlt_00769] ⌈

Service name:	BufferOverflowNotification		
Service ID [hex]:	`0x00000023`		
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
3	***overflowCounter***	`unit32`	Counter for the amount of lost Dlt messages since last sent MessageBufferOverflow message
Description:	The Dlt module sends this message when the dlt message buffer overflows. If not supported, NOT_SUPPORTED shall be the response.		

⌋ (RS_LT_00037)

5.3.18 SyncTimeStamp

[PRS_Dlt_00770] ⌈

Service name:	SyncTimeStamp		
Service ID [hex]	0x00000024		
:			
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	***synctimestamp***	TimeStamp	structure contains time stamp which represents the synchronized absolute time starting from 1970-01-01. Unit32 Nanoseconds part of the time Unit 32 Seconds part of the time Unit16 Seconds part of the time (MSB)
Description:	The Dlt module sends this message to report the synchronized absolute time. The message will be sent once when message transmission is started and then every 10 minutes. If not supported, NOT SUPPORTED shall be reported. The message shall also be sent after a jump in the time is detected, e.g. leap time or synchronization status.		

⌋ (RS_LT_00037)

5.3.19 Call SWC Injection

[PRS_Dlt_00217] ⌈CallSWCInjection messages shall be forwarded to the according application. The Service ID 0xFFF to 0xFFFFFFFF are reserved for this purpose. The value is user defined and can be freely used by an application. ⌋ ()

[PRS_Dlt_00218] ⌈In the case of a CallSWCInjection 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 identifies a unique client server interface of an application/runnable which is called in respect to reception of this message with the provided data. ⌋ ()

[PRS_Dlt_00219] ⌈If a unique identification is not possible (this pair does not exist, is not registered yet) the response shall be NOT_SUPPORTED. ⌋ ()

[PRS_Dlt_00220] ⌈

Service name:	CallSWCInjection		
Service ID [hex]:	`0x00000FFF ... 0xFFFFFFFF`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***dataLength***	`uint32`	length of the provided data
2	***data[]***	`uint8[]`	data to provide to the application
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 an application. If the Injection feature is disabled and/or not implemented, NOT_SUPPORTED shall be the response.		

⌋ ()

5.3.20 DLT Commands (deprecated)

[PRS_Dlt_00641]⌈ The following Dlt Commands are deprecated and not supported any more:

- 0x07 SetComInterfaceStatus
- 0x08 SetComInterfaceMaxBandwidth
- 0x09 SetVerboseMode
- 0x0C GetLocalTime
- 0x0D SetUseECUID
- 0x0E SetUseSessionID
- 0x0F SetUseTimestamp
- 0x10 SetUseExtendedHeader
- 0x14 MessageBufferOverflow
- 0x16 GetComInterfacelStatus
- 0x18 GetComInterfaceMaxBandwidth
- 0x19 GetVerboseModeStatus
- 0x1A GetMessageFilteringStatus
- 0x1B GetUseECUID
- 0x1C GetUseSessionID
- 0x1D GetUseTimestamp
- 0x1E GetUseExtendedHeader

⌋ (RS_LT_00002)

5.4 External Client / Tool

5.4.1 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.

[PRS_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 – 999.999)
12	4		ECU ID	Four characters the ECU ID
16	**Dlt log or trace message**			
			Header	
			Extended Header	
			Payload	

Table 5-14 Storage Header to store in front of a Dlt message.

⌋ (RS_LT_00002)

[PRS_Dlt_00427] ⌈The first entry in the Storage Header shall be a pattern 0 x 44 4C 54 01 (“DLT”+0x1). ⌋ (RS_LT_00002)

[PRS_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.⌋ (RS_LT_00002)

[PRS_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.⌋ (RS_LT_00002)

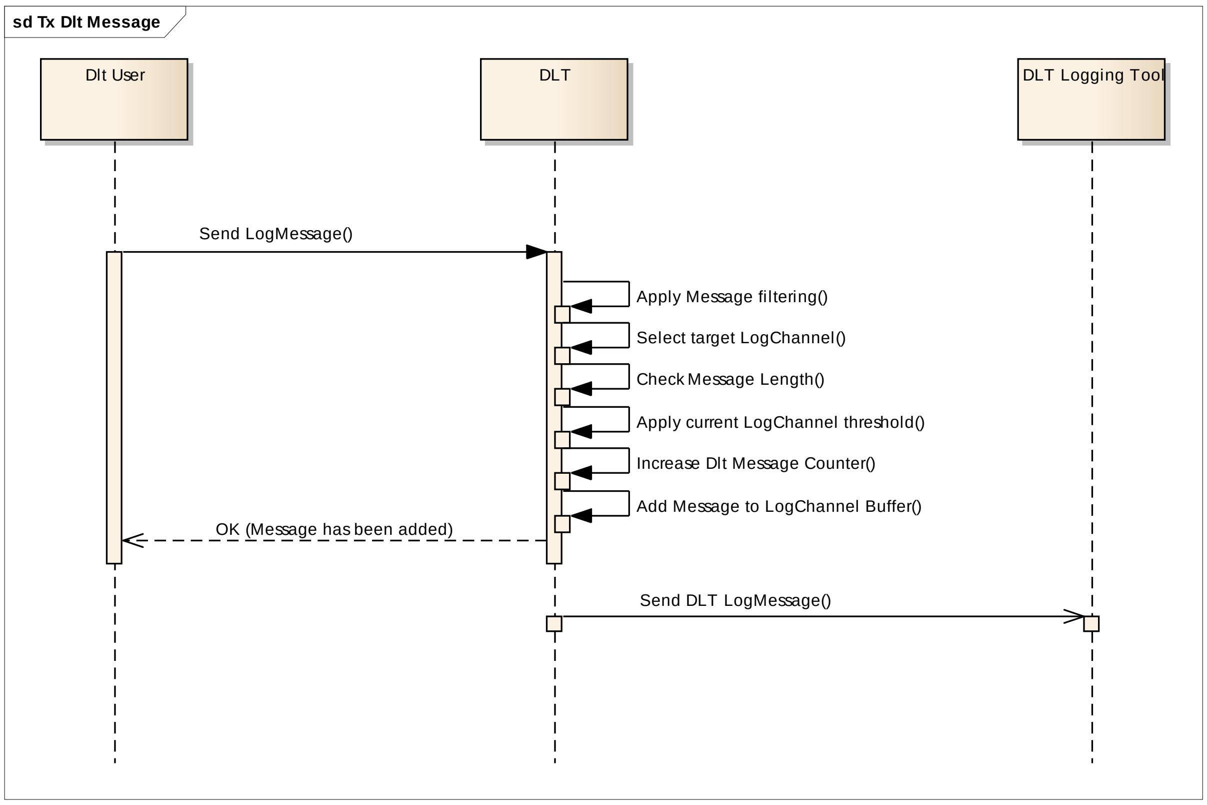
5.5 Sequences (lower layer)

5.5.1 States

N /A – The Dlt Protocol does not specify any states.

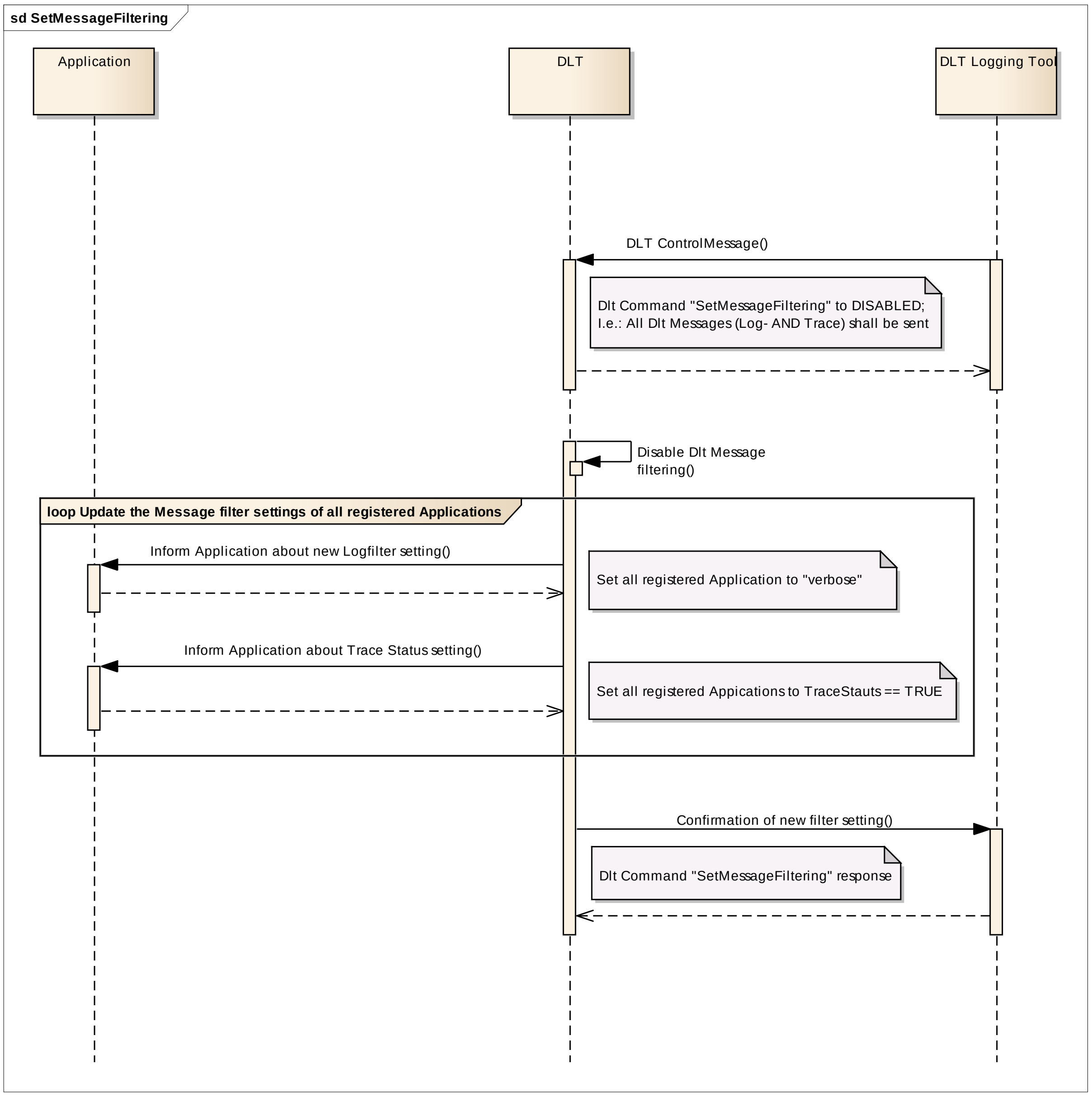
5.5.2 Control flow / Transitions

5.5.2.1 Transmission of Dlt Data Message


sd Tx Dlt Message
Dlt User
DLT
DLT Logging Tool
Send LogMessage()
Apply Message filtering()
Select target LogChannel()
Check Message Length()
Apply current LogChannel threshold()
Increase Dlt Message Counter()
Add Message to LogChannel Buffer()
OK (Message has been added)
Send DLT LogMessage()

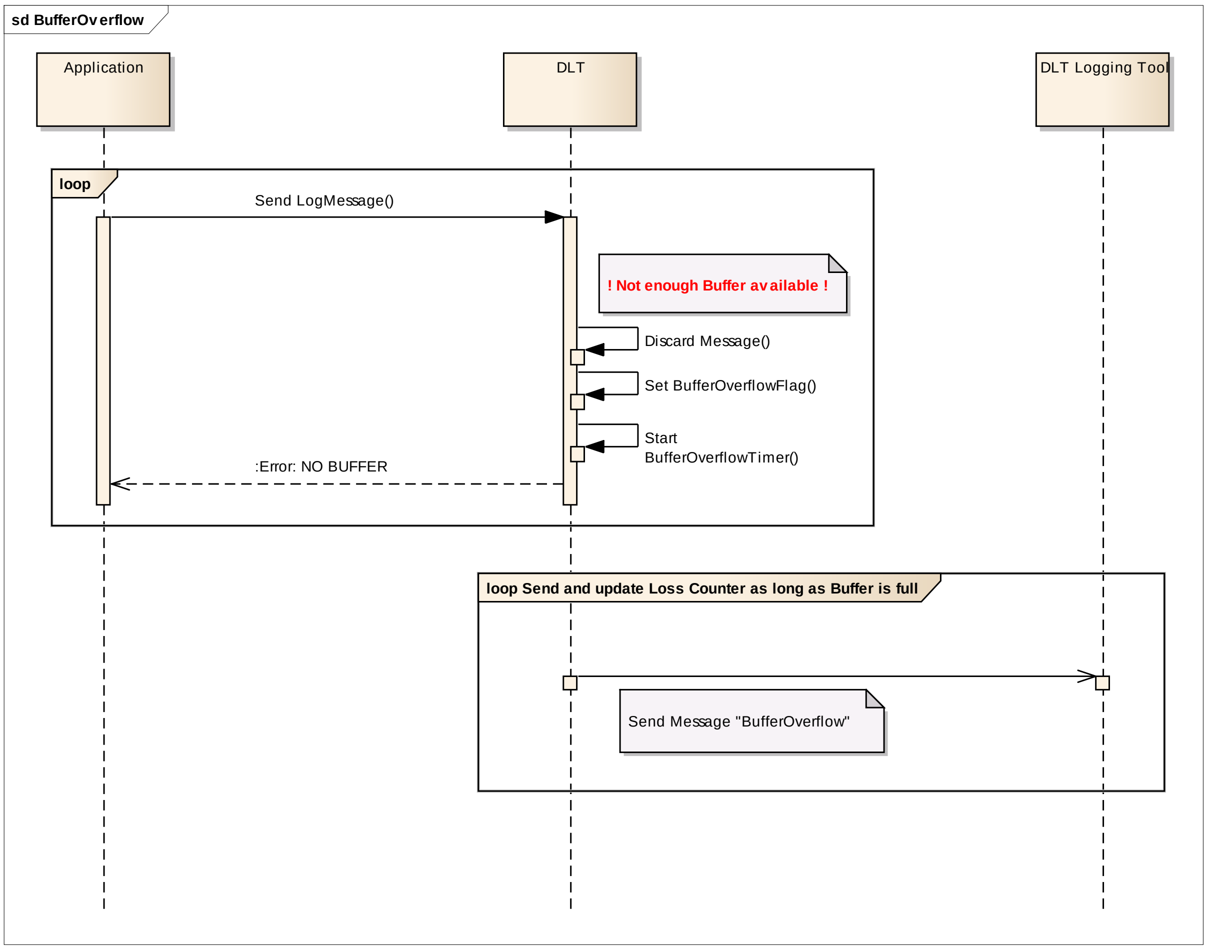
Sequence 1 – Transmission of Dlt Data Message

5.5.2.2 Set LogLevel Filter


sd SetMessageFiltering
Application
DLT
DLT Logging Tool
DLT ControlMessage()
Dlt Command "SetMessageFiltering" to DISABLED;
I.e.: All Dlt Messages (Log- AND Trace) shall be sent
Disable Dlt Message filtering()
loop Update the Message filter settings of all registered Applications
Inform Application about new Logfilter setting()
Set all registered Application to "verbose"
Inform Application about Trace Status setting()
Set all registered Appications to TraceStauts == TRUE
Confirmation of new filter setting()
Dlt Command "SetMessageFiltering" response

Sequence 2 - Set LogLevel Filter

5.5.2.3 Buffer Overflow


sd BufferOverflow
Application
DLT
DLT Logging Tool
loop
Send LogMessage()
! Not enough Buffer available !
Discard Message()
Set BufferOverflowFlag()
Start BufferOverflowTimer()
:Error: NO BUFFER
loop Send and update Loss Counter as long as Buffer is full
Send Message "BufferOverflow"

Sequence 3- Buffer Overflow

5.6 Error Handling

5.6.1 Error messages

5.6.1.1 Buffer Overflow

[PRS_Dlt_00648] ⌈If a Dlt Message Buffer Overflow occurs, the Control Message with the Service ID 0x23 (BufferOverflowNotification) shall be sent.⌋ ()

Note: The Service BufferOverflowNotification is defined in chapter 5..3.

[PRS_Dlt_00649] ⌈The status of the Dlt Message Buffer shall be cyclically checked.

The minimum time interval of sending the Dlt Overflow Message shall be configurable, i.e.: do not send more than one Dlt Overflow Message within the configured time span. ⌋ ()

5.6.1.2 Answering a Command with "ERROR"

[PRS_Dlt_00650] ⌈The Dlt module shall answer a Dlt Command with "ERROR" if one of the following cases:

- At least one of the received parameter values cannot be matched to the current configuration
- Another Dlt Command is currently in progress

⌋ ()

[PRS_Dlt_00642] ⌈If the Dlt module receives a Dlt command using a Service ID which is neither specified in chapter "Dlt Command" nor in chapter "Dlt Commands (deprecated)", the Dlt module shall answer with "ERROR". ⌋ ()

5.6.1.3 Answering a Command with "NOT SUPPORTED"

[PRS_Dlt_00644] ⌈The Dlt module shall respond with "DLT_NOT_SUPPORTED" if it receives one of the following Dlt Commands:

- 0x07 SetComInterfaceStatus
- 0x08 SetComInterfaceMaxBandwidth
- 0x09 SetVerboseMode
- 0x0A SetMessageFiltering
- 0x0C GetLocalTime
- 0x0D SetUseECUID
- 0x0E SetUseSessionID

- 0x0F SetUseTimestamp
- 0x10 SetUseExtendedHeader
- 0x14 MessageBufferOverflow
- 0x16 GetComInterfacelStatus
- 0x18 GetComInterfaceMaxBandwidth
- 0x19 GetVerboseModeStatus
- 0x1A GetMessageFilteringStatus
- 0x1B GetIseECUID
- 0x1C GetUseSessionID
- 0x1D GetUseTimestamp
- 0x1E GetUseExtendedHeader

⌋ ()

5.6.2 Error resolution

5.6.2.1 Transmission Retry

[PRS_Dlt_00651] ⌈In case an error occurred while trying to send a Dlt Message on the bus, the Dlt module shall re-try to send it. The maximum amount of transmission retries shall be configurable. ⌋ ()

Note:
This is not part of the Dlt Protocol itself, but recommended for the implementation of the Dlt Module.

6 Protocol usage and guidelines

6.1 Proposal for usage of Log Levels

The log levels as defined in 5.1.1.2.1 by PRS_Dlt_00619 bear an implied semantics. This section gives a recommendation on how to use different log levels, i.e. which kind of event should be reported by which log level. However, this section is purely informational. That is, the log message producer SHOULD comply to this section but MAY choose to imply a different semantics. Also, the log message consumer SHALL NOT derive actions from messages of certain log levels without additional agreement (e.g. by negotiating a common profile by different means).

6.1.1 Log Level FATAL (DLT_LOG_FATAL)

Fatal, unrecoverable error. Accordingly, these messages should occur on very rare occasions. The whole (sub-)system's stability might be endangered. Should be used before a (sub-)system enters a failsafe state (e.g. emergency shutdown) or if it encounters an error that will most likely cause an imminent crash. Often the last message a (sub-)system can log.
Examples:

- a corrupted boot environment
- a hardware component that is vital for (sub-)system startup fails or is missing
- a critical application, service or other software component exited unexpectedly
- may also be used if an application exits due to a fatal error

6.1.2 Log Level ERROR (DLT_LOG_ERROR)

Errors denote conditions that will cause the (sub-)system to stop working correctly but that might be recoverable.
Examples:

- missing or failing non-vital services, applications or other software components
- hardware on which an application depends is inaccessible
- a network connection that is required for correct functionality is failing
- a required file is missing, inaccessible or corrupted

6.1.3 Log level WARNING (DLT_LOG_WARNING)

Used if correct behavior cannot be ensured. Something that's concerning but not causing the operation to abort. The condition might become a problem in the future resulting in an error, or might not. Runtime situations that are undesirable, unexpected and potentially lead to an error or cause application oddities, but everything still under control. Automatic recovery from the situation exists (e.g. a handled exception) and the application can continue. Warnings may give hints at the root cause of subsequent errors.

Examples:

- bad login attempts
- unexpected data during import jobs
- switching from a primary to backup server
- short loss of network or database connectivity as long as it does not inevitably result in faulty behavior
- number of resources in a pool getting low
- an unusual-but-expected timeout in an operation
- not enough disk space for a core dump
- processes exceed their maximum execution time as per specification

6.1.4 Log level INFO (DLT_LOG_INFO)

Should give an overview of major state changes providing high level context for understanding any warnings or errors that also occur. It can be used on runtime events that are normal but somehow important.
Examples:

- key system and hardware information on startup / shutdown
- startup / shutdown of an application
- successful initialization
- a long running job is starting and ending
- successful completion of significant transactions
- entries and exits from key areas of an application
- external devices connected / detached (e.g. USB drives)
- errors or connection loss of connected devices that do not affect the system's stability (e.g. mobile phones, multimedia devices, ...)
- information vital for determining key performance indicators (KPI)

6.1.5 Log level DEBUG (DLT_LOG_DEBUG)

Fine-grained debug-level messages. Detailed, diagnostically helpful, information for programmers, normally of use only when debugging a program. This and below log levels should only be used for development and testing and disabled for production systems.
Examples:

- entry and exit points into functions
- function parameters passed
- values, value changes or state changes of key variables, usually not complete array dumps though
- return values
- information about received events
- network connection information
- debugging information of connected hardware
- other significant information to reconstruct the flow through the system

6.1.6 Log level VERBOSE (DLT_LOG_VERBOSE)

Even more fine-grained information than DEBUG. Should be used for in-depth debug information.
Examples:

- dump of buffers, arrays or memory segments
- information about loops and iterations
- detailed network information
- detailed hardware state information

6.2 Support for ECU-, Application- and Context-ID of arbitrary length

The general approach is to transmit IDs of arbitrary length as payload arguments and to use the regular keys as references to the correct payload argument.

6.2.1 Description of solution

The description of the ECU ID, Application ID and Context ID (i.e. “long IDs”) will be transmitted within the first three payload arguments as regular strings. The DLT user SHALL transmit them before every regular payload argument and SHALL send them in the order ECU ID, Application ID, Context ID. Every combination of regular 4-Byte-IDs (without payload argument) and Long IDs SHALL be possible.

If at least one long ID is transmitted, the corresponding short ID serves three purposes:

1. The short IDs indicates to the client (e.g. the DLT-Viewer or any other renderer) that this scheme is used and a corresponding ID is present in the payload.
2. It references the correct payload field so the client can interpret the field(s) as IDs.
3. Because the DLT protocol and its implementation is not aware of long IDs, the internals will still use the regular short IDs. As such, collisions need to be avoided in that an ECU may not register an Application ID twice and an application may not register a Context ID twice.

The next section explains how valid short IDs are selected (or rather constructed).

6.2.2 Valid IDs

Table 6-1 shows the new semantics of the contents of an Application ID or a Context ID. Byte 0 and byte 1 contain fixed values. They represent a magic sequence that is unlikely to be found in real ECUs and indicate to the receiver that this scheme is employed. The remaining two bytes (2 and 3) consist of bits that take care of collision avoidance (CA), a payload field indicator (PI) and fixed bits (--).

Name	ECU ID (Standard Header), Application ID or Context ID (Extended Header)																	
Byte	0	1	2								3							
Bit-No	--	--	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Function	Magic sequence		CA		--	CA					CA		--	CA		PI		CA
Value	0x7D	0x5B	*	*	1	*	*	*	*	*	*	*	1	*	*	*	*	*

Table 6-1 Contents of 4-byte-IDs when referencing long IDs in DLT payload

The payload field indicator PI contains the reference to the correct payload argument and is to be interpreted as the offset from the first payload argument. Collision avoidance is necessary because Application IDs need to be unique* within one ECU and Context IDs have to be unique* within one application context. As before, this is the responsibility of the developer, integrator or the implementation.

** as in: they may not be registered twice.*

6.2.3 Message format

A complete message will therefore consist of

- The Standard DLT Header
- The Extended DLT Header
- 1 – 3 string payload arguments
- The actual payload arguments

When long IDs are to be transmitted, the message SHALL contain the Extended Header and the corresponding flags need to be set.
Long IDs are transmitted as regular string arguments. This means, the payload argument shall contain a Type Info field with the values 0x00020000 for an ASCII-coded string or 0x00020100 for a UTF-8-coded string according to PRS_Dlt_00625 and PRS_Dlt_00627. In compliance with PRS_Dlt_00373, the Type Info field is followed by the string length of the long ID including the terminating \0-byte, encoded as an unsigned 16-bit integer, and the null-terminated data string.

7 Configuration specification

This chapter lists all parameter the Dlt Protocol uses.

7.1 Dlt Header

Long Name	Short Name	Description
Header Type	HTTY	Metainformation about the Dlt Header
Message Counter	MCNT	Message counter of Dlt Tx messages
Length	LEN	Length of Dlt Message
ECU ID (optional)	ECU	Name of ECU
Session ID (optional)	SEID	Session ID
Timestamp (optional)	TMSP	Timestamp

7.1.1 Header Type

Long Name	Short Name	Description
Use Extended Header	UEH	Flag for usage of Dlt Extended Header
MSB First	MSBF	Flag for Little/Big endianness
With ECU ID	WEID	Flag for ECU ID field usage
With Session ID	WSID	Flag for Session ID field usage
With Timestamp	WTMS	Flag for Timestamp field usage
Version number	VERS	Contains the used Dlt Protocol Version

7.2 Dlt Extended Header

Long Name	Short Name	Description
Message Info	MSNI	Metainformation about the Extended Header
Number of Arguments	NOAR	Number of arguments contained in payload
Application ID	APID	Application ID
Context ID	CTID	Context ID

7.2.1 Message Info

Long Name	Short Name	Description
Verbose mode	VERB	Flag to indicate the usage of verbose mode
Message Type	MSTP	Identification of the type of Dlt message
Message Type Info	MTIN	Additional information of the message type

7.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the protocol is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

Additional module-specific published parameters are listed below if applicable.

- AUTOSAR confidential -