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	R22-11

Document Change History

Date	Release	Changed by	Description
2022-11-24	R22-11	AUTOSAR Release Management	• Context IDs for the framework reserved • Optional fragmentation (segmentation) header allowed • Support of long Appl-IDs and Context-IDs added
2021-11-25	R21-11	AUTOSAR Release Management	• Add support for specific format and precision coding (TYFM & TYPR). • Fix "RestoreToFactoryDefault()" to "ResetToFactoryDefault()" and fixes in response parameter of "GetLogInfo()". • Introduction of v2 of the protocol. • Non-VerboseMessage-format-ARXML is now in TPS_LogAndTrace_Extract.
2020-11-30	R20-11	AUTOSAR Release Management	• Restructured document for better differentiation between verbose and non-verbose mode • Improved definition of “first” DLT arguments • Reworked Use Case diagrams • Fixed contradicting message counter requirements
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

Document Change History

Date	Release	Changed by	Description
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

1 Introduction and overview

This protocol specification defines 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 of 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 to the Dlt module.

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 non-volatile (if supported by hardware).

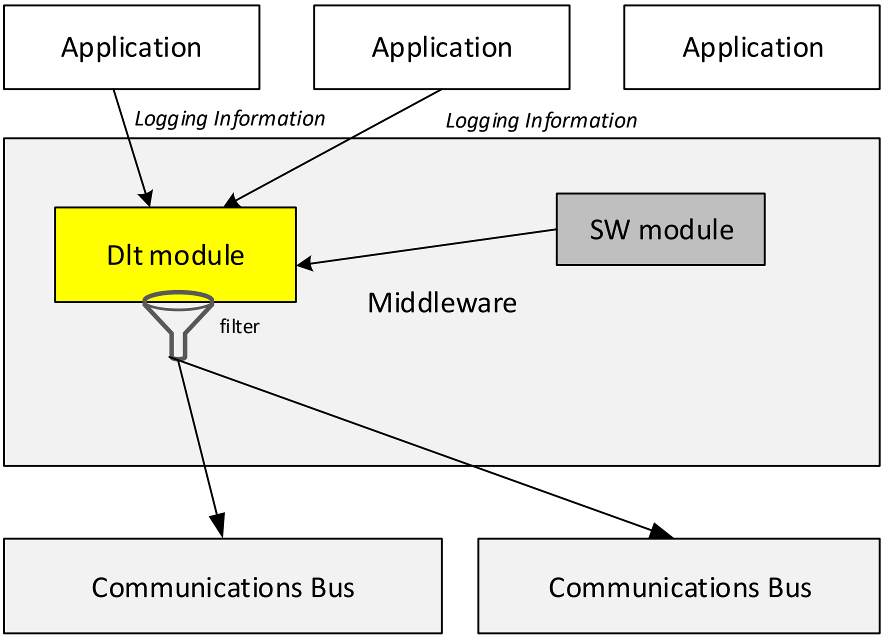


Figure 2 – Location of the Dlt Protocol

While the version "1" of the protocol still keeps its validity (look up [2], Log and Trace Protocol Specification from R20-11), version "2" of the protocol, as specified in this document, increases flexibility for the required use cases and introduces new features, which can't be anymore added to the previous protocol version in a backward-compatible way. These are:

- Tags for message filtering and Privacy Flags.
- Improved Timestamp with a nanosecond resolution and a possible time span of thousands of years. In connection with that, the ServiceID / command 0x24 / "SyncTimeStamp" is no more needed and got removed.

- Long IDs for ECU, Application and Context names are now already supported in the header section of the Log and Trace message.
- The same applies for a possible source file name and line number information.

The used "flag-length-value" approach leaves room for future extensions of the protocol without breaking the compatibility with current features.

Additionally, strings in the Log and Trace message do not need any more a null-termination, since the length value for a string is already enough.

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. See also [1];

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 and documents

[1] ISO/IEC 7498-1:1994 Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model - Part 1
www.iso.org

[2] Log and Trace Protocol Specification v.1
AUTOSAR_PRS_LogAndTraceProtocol from release R20-11

[3] AUTOSAR Log And Trace Extract Template
AUTOSAR_TPS_LogAndTraceExtract

1.4 Dependencies to the Application Layer

To transmit Dlt messages, the applications need to know whether to send the Dlt messages using the verbose 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 Usage example: General logging with Dlt

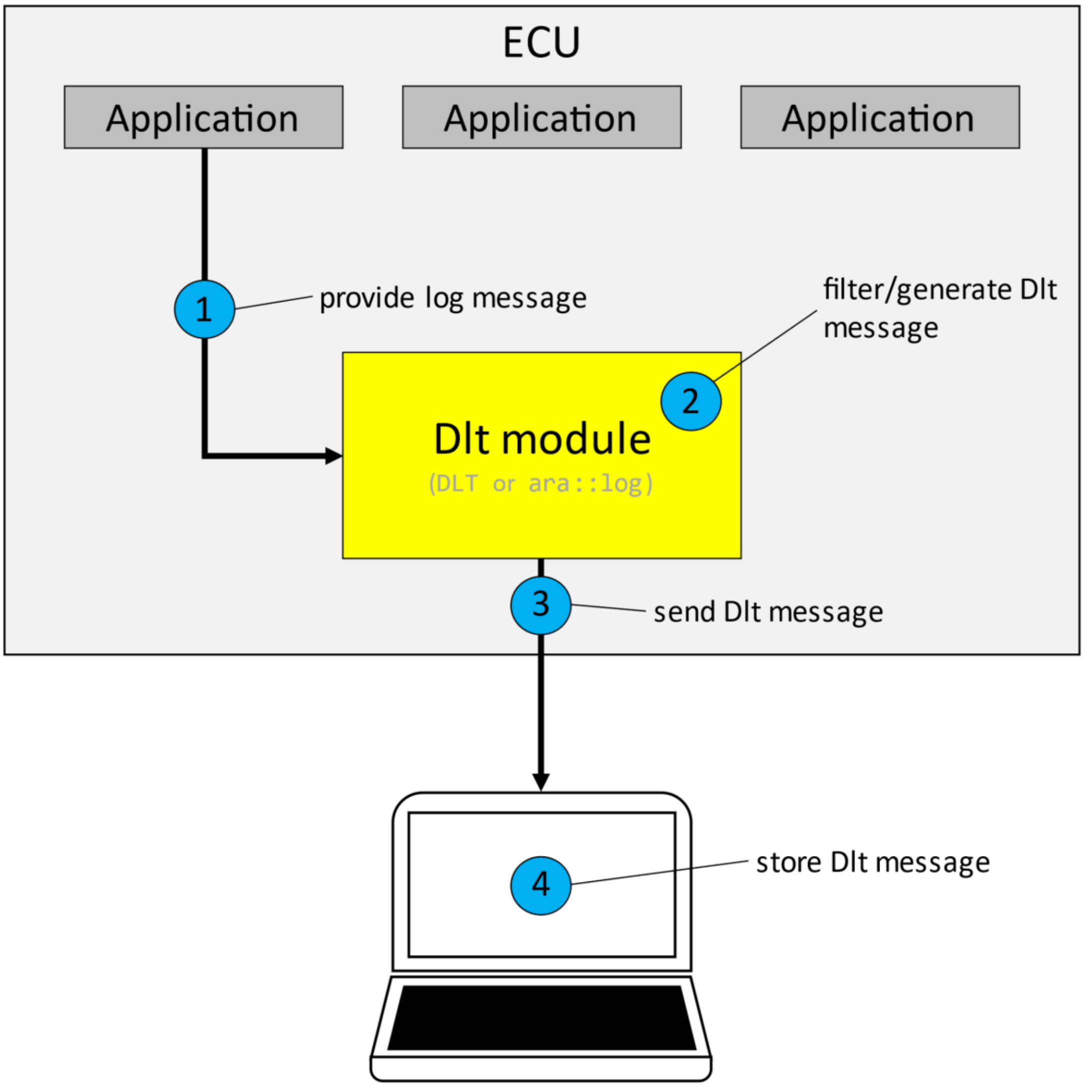


Figure 3: General logging with Dlt

(1) An application/SW-C is providing a log message to the Dlt module.
(2) The log message is either filtered or a Dlt message is created by the Dlt module which implements the Dlt Protocol. (Depending on log level.)
(3) The Dlt module sends the Dlt message to the communications bus.
(4) An external client receives and stores the Dlt message.

2.1.2 Usage example: Tracing of VFB

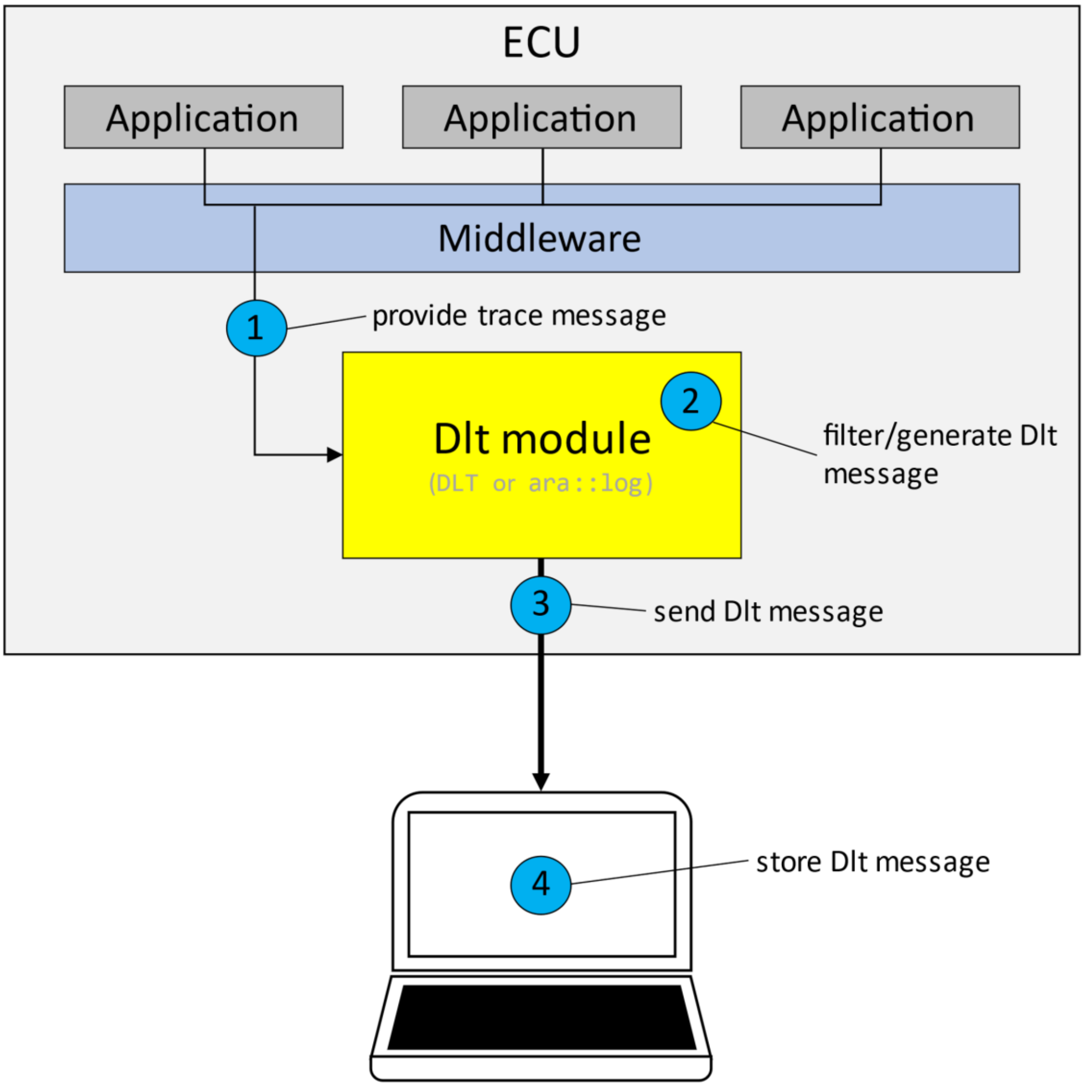


Figure 4: Tracing of VFB

(1) Middleware calls the macro provided by Dlt module 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 Usage example: Runtime configuration of Dlt

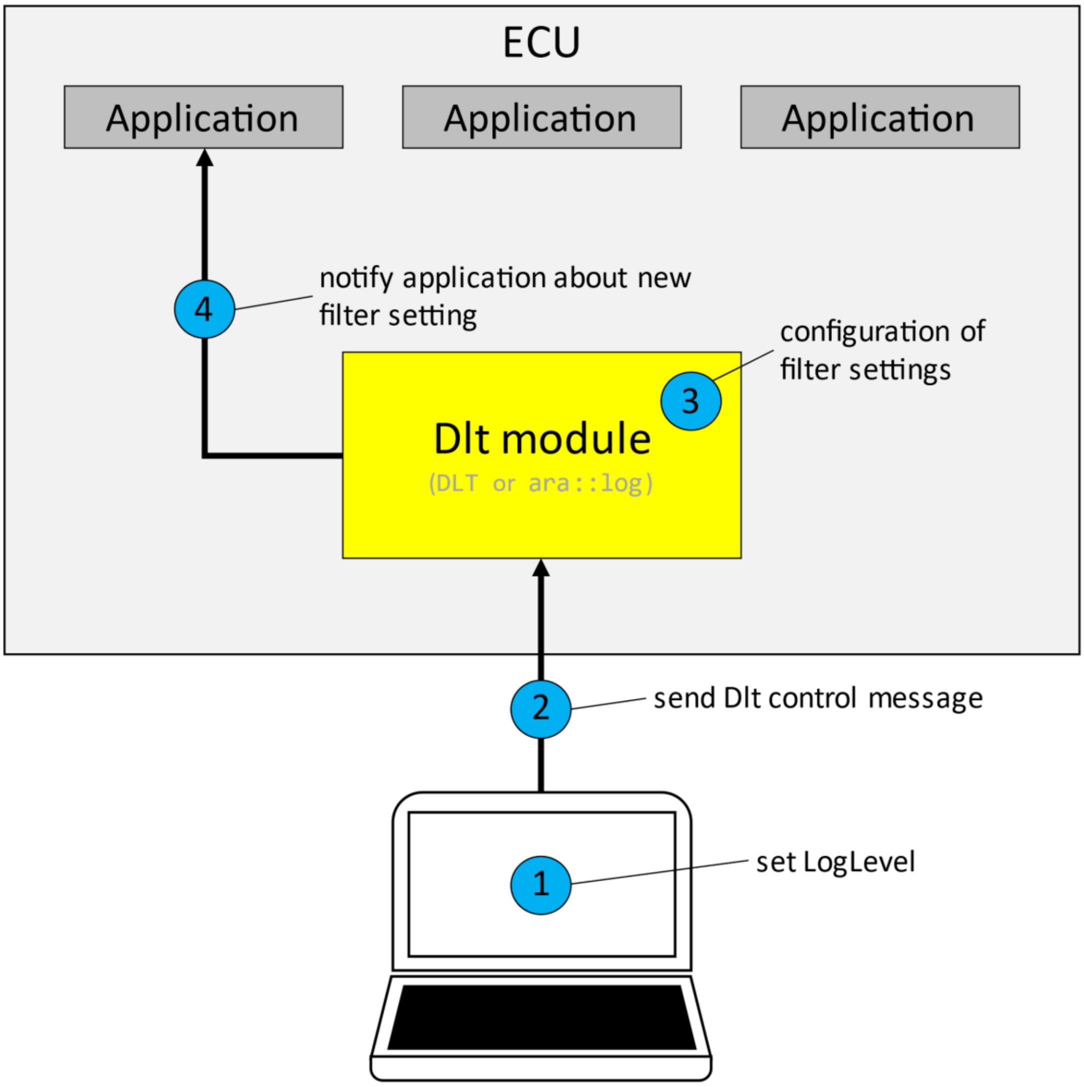


Figure 5: Runtime configuration of Dlt

(1) An external client sets the log and trace level and sends the changes to the Dlt module which implements the Dlt protocol.
(2) Via a Dlt control message the change is sent to the Dlt module which implements the Dlt protocol.
(3) The Dlt module adapts its configuration of filter settings accordingly.
(4) The Dlt module informs the application about the new log level.

2.1.4 Usage example: 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 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.

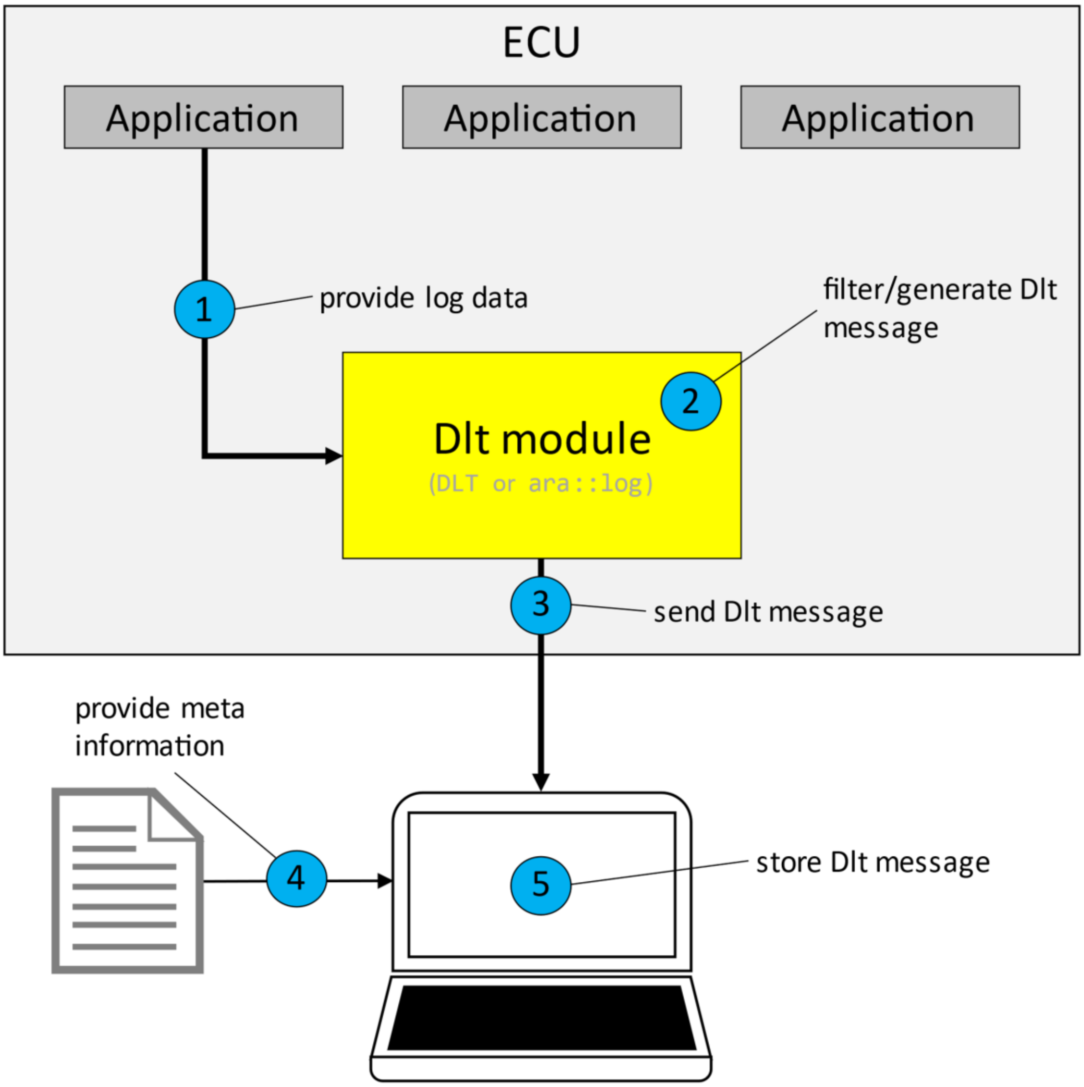


Figure 6: Logging in non-verbose mode

(1) An application/SW-C is providing non-verbose logging data to the Dlt module.
(2) The Dlt module filters and generates the Dlt message.
(3) The Dlt module sends the Dlt message to the communications bus.
(4) An external client fetches meta information from an external file.
(5) The merged information is stored by an external 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_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_00614, PRS_Dlt_00618, PRS_Dlt_00619, PRS_Dlt_00620, PRS_Dlt_00621, PRS_Dlt_00622, PRS_Dlt_00641, PRS_Dlt_01001, PRS_Dlt_01002, PRS_Dlt_01003, PRS_Dlt_01004, PRS_Dlt_01005, PRS_Dlt_01006, PRS_Dlt_01007, PRS_Dlt_01008, PRS_Dlt_01009, PRS_Dlt_01010, PRS_Dlt_01011, PRS_Dlt_01015, PRS_Dlt_01016
RS_LT_00013	The transmitted data shall be packetized.	PRS_Dlt_00314, PRS_Dlt_00315, PRS_Dlt_00409, PRS_Dlt_01000, PRS_Dlt_01001, PRS_Dlt_01002, PRS_Dlt_01003, PRS_Dlt_01004, PRS_Dlt_01005, PRS_Dlt_01043, PRS_Dlt_01044, PRS_Dlt_01045, PRS_Dlt_01046, PRS_Dlt_01048, PRS_Dlt_01049, PRS_Dlt_01050, PRS_Dlt_01051
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_01000
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_01004, PRS_Dlt_01012, PRS_Dlt_01013, PRS_Dlt_01014
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, PRS_Dlt_01046, PRS_Dlt_01048, PRS_Dlt_01050
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_01024
RS_LT_00021	There shall be a logical grouping for log messages by using different identifiers.	PRS_Dlt_01020, PRS_Dlt_01021, PRS_Dlt_01022, PRS_Dlt_01023, PRS_Dlt_01031, PRS_Dlt_01032, PRS_Dlt_01033, PRS_Dlt_01034, PRS_Dlt_01035, PRS_Dlt_01036
RS_LT_00022	Each ECU shall have its unique ECU ID.	PRS_Dlt_01017, PRS_Dlt_01018, PRS_Dlt_01019
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_01037, PRS_Dlt_01052
RS_LT_00024	It shall be possible to transmit the parameters in a raw format.	PRS_Dlt_00126
RS_LT_00025	It shall be possible to transmit ASCII text in log or	PRS_Dlt_00156, PRS_Dlt_00157, PRS_Dlt_00373, PRS_Dlt_00410, PRS_Dlt_00420, PRS_Dlt_00782,

	trace messages.	PRS_Dlt_00783, PRS_Dlt_00786, PRS_Dlt_00787, PRS_Dlt_00790, PRS_Dlt_00791, PRS_Dlt_00793, PRS_Dlt_00794, PRS_Dlt_00803, PRS_Dlt_01038, PRS_Dlt_01039
RS_LT_00026	The data in non-verbose mode shall be described by an extra file.	PRS_Dlt_00134, PRS_Dlt_00353, PRS_Dlt_01037, PRS_Dlt_01052
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_00624
RS_LT_00030	Logging shall be able to monitor and shape the amount of LT log and trace events.	PRS_Dlt_00651
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_00187, PRS_Dlt_00194, PRS_Dlt_00195, PRS_Dlt_00196, PRS_Dlt_00197, PRS_Dlt_00198, PRS_Dlt_00199, PRS_Dlt_00200, PRS_Dlt_00217, PRS_Dlt_00218, PRS_Dlt_00219, PRS_Dlt_00220, 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, PRS_Dlt_00642, PRS_Dlt_00644, PRS_Dlt_00650, PRS_Dlt_01040, PRS_Dlt_01041, PRS_Dlt_01042, PRS_Dlt_01057, PRS_Dlt_01058, PRS_Dlt_01059, PRS_Dlt_01060, PRS_Dlt_01061
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_00648, PRS_Dlt_00649, PRS_Dlt_00769
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	Logging shall be able to handle raw buffer content as logging information.	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_00172, PRS_Dlt_00173, PRS_Dlt_00175, PRS_Dlt_00176, PRS_Dlt_00177, 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_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_00412,

		PRS_Dlt_00414, PRS_Dlt_00422, PRS_Dlt_00423, PRS_Dlt_00459, PRS_Dlt_00625, PRS_Dlt_00626, PRS_Dlt_00791, PRS_Dlt_00794, PRS_Dlt_00803
RS_LT_00056	There shall be the possibility to transmit the parameters with additional information about themselves (self-description).	PRS_Dlt_00782, PRS_Dlt_00783, PRS_Dlt_00784, PRS_Dlt_00785, PRS_Dlt_00786, PRS_Dlt_00787, PRS_Dlt_00788, PRS_Dlt_00789, PRS_Dlt_00790, PRS_Dlt_00791, PRS_Dlt_00792, PRS_Dlt_00793, PRS_Dlt_00794, PRS_Dlt_00795, PRS_Dlt_00796, PRS_Dlt_00797, PRS_Dlt_00798, PRS_Dlt_00799, PRS_Dlt_00800, PRS_Dlt_00801, PRS_Dlt_00802, PRS_Dlt_00803, PRS_Dlt_01025, PRS_Dlt_01026, PRS_Dlt_01027, PRS_Dlt_01028, PRS_Dlt_01029, PRS_Dlt_01030
RS_LT_00058	AUTOSAR Log and Trace shall support harmonized logging.	PRS_Dlt_01053, PRS_Dlt_01054, PRS_Dlt_01055, PRS_Dlt_01056, PRS_Dlt_01062

4 Acronyms and abbreviations

Abbreviation / Acronym	***Description***
APID	Application ID
CNTI	Content information
CTID	Context ID
Dlt	Diagnostic Log and Trace
HTYP	Header Type
LEN	Overall Message Length
MCNT	Message Counter
MSID	Message ID
MSIN	Message Info
MSTP	Message Type
MTIN	Message Type Info
NOAR	Number of Arguments
TMSP	Timestamp
VERB	Verbose
VERS	Version Number
WACID	With App- and Context ID
WEID	With ECU ID
WPVL	With Privacy Level
WSFLN	With Source File Name and Line Number
WSID	With Session ID
WSGM	With Segmentation
WTGS	With Tags

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 8-bit ASCII characters (e.g. "ABS0" or "InstrumentCluster").
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 the name (or the abbreviation of it) of a SW-C / application. It identifies the SW-C / application that 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”. 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 and is used in conjunction with the "Non-Verbose Mode" (see later).
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 sent.
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.
string	For the Log and Trace Protocol a "string" defines a sequence of characters. Its length is either predefined by this document here, or there is a length specifier in front of it. The string contains only the character data for the text it represents without a special terminating item like the NUL-character (\0).

Verbose Mode	When the Dlt module is sending out Log and Trace information onto the communication channel, the sent data message contains also predefined, static data elements like names, units, format information, length and so on;
Non-Verbose Mode	To reduce the network load on the communication channel, the Log and Trace data message contains as little of such metadata as possible. One of them is the Message-ID, which uniquely identifies this message and all of its content. All the above mentioned meta-data is handed over to the receiver via a separate extract file: together with the Message-ID, the receiver of a Non-Verbose Mode message can reassemble the complete content, like sent in a Verbose Mode message.

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 "Base Header", an optional "Extension Header", and a Payload segment.

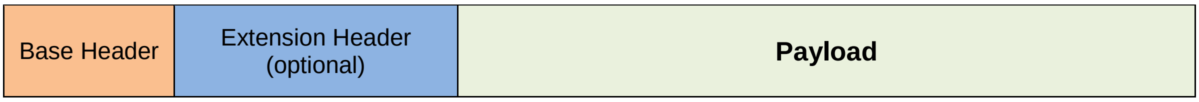
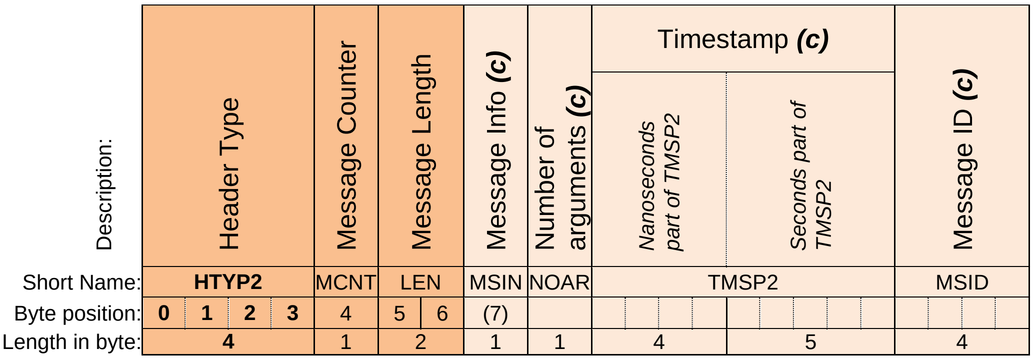


Figure 6 - Dlt message format

[PRS_Dlt_01000] ⌈The "Base Header" and the "Extension Header" shall always use the network byte order.⌋ (RS_LT_00016,RS_LT_00013)

Note: "Network Byte Order" equals "Big Endian".

5.1.1 Base Header



Description:	Header Type	Message Counter	Message Length	Message Info **(c)**	Number of arguments **(c)**	Timestamp **(c)**		Message ID **(c)**
						Nanoseconds part of TMSP2	*Seconds part of TMSP2*	
Short Name:	**HTYP2**	MCNT	LEN	MSIN	NOAR	TMSP2		MSID
Byte position:	**0** **1** **2** **3**	4	5 6	(7)				
Length in byte:	4	1	2	1	1	4	5	4

Figure 7 - Base Header; *(c): conditional*

[PRS_Dlt_01001] ⌈The Base Header shall always consist of the following fields in the following order:

- Byte 0 – 3: HTYP2 (Header Type for protocol version "2")
- Byte 4: MCNT (Message Counter)
- Byte 5 – 6: LEN (Message Length)

⌋(RS_LT_00002, RS_LT_00013)

The following fields of the Base Header are only contained if certain conditions are met. The conditions are defined later in this sub-chapter.

[PRS_Dlt_01002] ⌈In addition, the Base Header shall also conditionally consist of the following fields in the following order:

• MSIN	(Message Info)	length; 1 byte;
• NOAR	(Number of arguments)	length; 1 byte;
• TMSP2	(Timestamp version "2")	length; 9 bytes;
• MSID	(Message ID)	length; 4 bytes;

Each element shall be added after the last existing element in the Base Header. There shall be no gap in between.⌋ (RS_LT_00002, RS_LT_00013)

Note: Since the above elements are conditional, an absolute byte position can't be given here, as they may shift due to the activation/deactivation of those conditional fields (see above).

[PRS_Dlt_01003] ⌈If the Log and Trace message is a Data Message in Verbose Mode or a Control Message, the MSIN (Message Info) and NOAR (Number of arguments) shall be added to the Base Header.⌋ (RS_LT_00002, RS_LT_00013)

Note: The information, whether the Log and Trace message is a Data Message in Verbose or Non-Verbose Mode or a Control Message, is located in the HTYP2 – field sub-element "CNTI" (Content Info); see the following subchapter for more details.

[PRS_Dlt_01004] ⌈If the Log and Trace message is a Data Message (Verbose Mode or Non-Verbose Mode), the TMSP2 (Timestamp) with a nanosecond resolution shall be added to the Base Header.⌋ (RS_LT_00002, RS_LT_00013, RS_LT_00017)

[PRS_Dlt_01005] ⌈If the Log and Trace message is a Non-Verbose Mode Data Message, the MSID (Message ID) shall be added to the Base Header.⌋
(RS_LT_00002, RS_LT_00013)

5.1.1.1 Header Type

The "Header Type"-field for protocol version '2' (HTYP2) contains general information about the Log and Trace message.
Except for the three bits "Version Number" information, all other flags are used to indicate conditional or optional later content in this Log and Trace message.
In this context here,

- "**conditional**" means, the required usage is specified in this document.
- "**optional**" means, the usage is application specific.

[PRS_Dlt_01006] ⌈The Header Type - field (HTYP2) shall be the first element of any Log and Trace message.⌋ (RS_LT_00002)

[PRS_Dlt_01007] ⌈The size of the Header Type - field (HTYP2) shall be 32 bit.⌋ (RS_LT_00002)

[PRS_Dlt_01008] ⌈The Header Type (HTYP2) shall contain the following information and shall be encoded in the following way:

Bit 0 – 1:	CNTI	(Content Information)
Bit 2:	WEID	(With ECU ID)
Bit 3:	WACID	(With App- and Context ID)
Bit 4:	WSID	(With Session ID)
Bit 5-7:	VERS	(Version Number)
Bit 8:	WSFLN	(With Source File Name and Line Number)
Bit 9:	WTGS	(With Tags)
Bit 10:	WPVL	(With Privacy Level)
Bit 11	WSGM	(With Segmentation)
Bit 12 - 31:	reserved	(reserved by AUTOSAR for future usage)

⌋(RS_LT_00002)

	Header Type (HTYP2)							
Byte Bit	7	6	5	4	3	2	1	0
0	**VERS**			**WSID**	**WACID**	**WEID**	**CNTI**	
1	reserved	reserved	reserved	reserved	**WSGM**	**WPVL**	**WTGS**	**WSFLN**
2	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
3	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved

Figure 8 - Encoding of Header Type

[PRS_Dlt_01009] ⌈The two "CNTI"-bits (Content Info; bits 0 – 1 in HTYP2) shall be a 2-bit unsigned integer and shall be encoded in the following way:

- 0x0: Verbose Mode Data Message;
- 0x1: Non-Verbose Mode Data Message;
- 0x2: Control Message;
- 0x3: reserved;

⌋(RS_LT_00002)

[PRS_Dlt_01010] ⌈The "VERS"-bits (Version Number; bits 5 – 7 in HTYP2) shall be a 3-bit unsigned integer and shall contain the Log and Trace protocol version as defined by AUTOSAR. The version number valid for this specification release is "2".⌋ (RS_LT_00002)

Note: The "VERS"-bits are located at the same position like in version "1" of the protocol. Therefore the receivers can always distinguish the protocol versions.

[PRS_Dlt_01011] ⌈If one of the following bits are set, the "Extension Header" shall be added after the "Base Header":

Bit 2:	WEID	(With ECU ID)
Bit 3:	WACID	(With App- and Context ID)
Bit 4:	WSID	(With Session ID)
Bit 8:	WSFLN	(With Source File Name and Line Number)
Bit 9:	WTGS	(With Tags)
Bit 10:	WPVL	(With Privacy Level)
Bit 11:	WSGM	(With Segmentation)

⌋(RS_LT_00002)

Note: The details about the "Extension Header" and the correlation with the above-mentioned bits are specified in a later subchapter.
Also, the bits 12 – 31 (currently "reserved by AUTOSAR for future usage") are intended to require the "Extension Header" in the future.

5.1.1.2 Message Counter

The Message Counter (MCNT) counts Dlt messages transmitted to a selected Log Channel. Each Log Channel needs to maintain its own Message Counter. On the receiver side, the Message Counter value can be evaluated to identify lost messages 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 Message Counter shall be incremented by one for each Dlt message that is transmitted to assigned LogChannel.⌋ (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)

5.1.1.3 Message Length

[PRS_Dlt_00320] ⌈The Message Length (LEN) field for the complete Log and Trace message in the Base Header shall be a 16-bit unsigned integer.⌋ (RS_LT_00002)

[PRS_Dlt_00614] ⌈The Message Length (LEN) field in the Base Header shall be set to the overall length in bytes of the complete Log and Trace message, which is the sum of:

- the length in bytes of the Base Header itself,
- the length in bytes of the optional Extension Header and
- the length in bytes of the optional Payload.

⌋ (RS_LT_00002)

Note: This Message Length (LEN) contains the length of a single simple LogAndTraceMessage and is independent from any segmentation functionality, as specified later on (compare chapter "5.1.2.7 Optional Message Segmentation").

Therefore, the upper limit of a single simple LogAndTraceMessage is either limited by the underlying communication protocol / -medium or by the max.value of the LEN field (16bit): 65535.

5.1.1.4 Conditional "Message Info"

Like specified above (refer PRS_Dlt_01003), the MSIN (Message Info) is added to the Base Header in case the Log and Trace message is a Data Message in Verbose Mode or a Control Message, otherwise the MSIN is not part of the Base Header.

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

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

⌋ (RS_LT_00002)

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

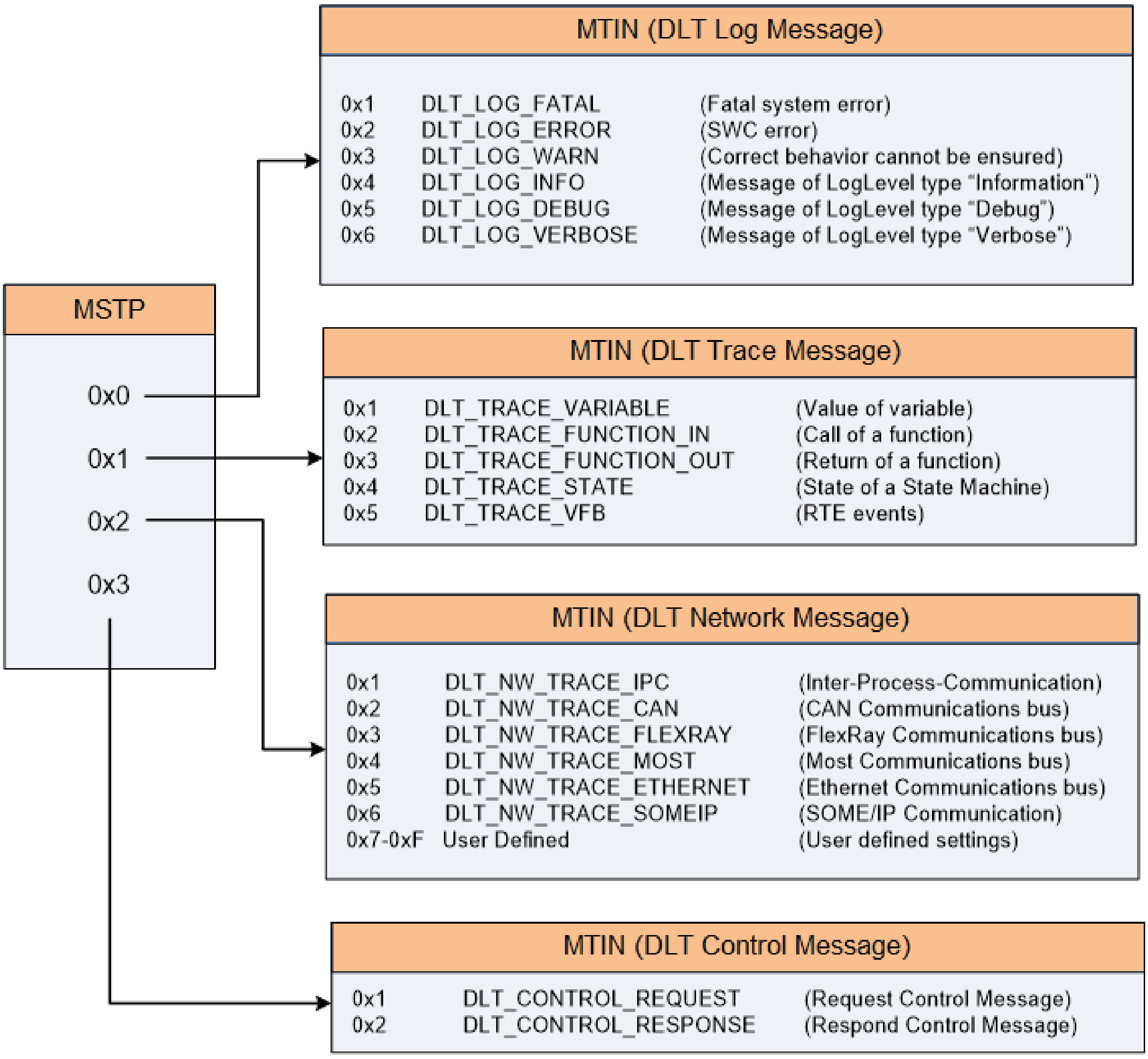
Figure 10 - Encoding of the Message Info field

[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)



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] ⌈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)

5.1.1.5 Conditional "Number of Arguments"

Like specified above (refer PRS_Dlt_01003), the NOAR (Number of Arguments) is added to the Base Header in case the Log and Trace message is a Data Message in Verbose Mode or a Control Message. Otherwise the NOAR is not part or the Base Header.

Number of Arguments represents the number of consecutive parameters or the number of consecutive control commands 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] ⌈ The Number of Arguments field (NOAR) shall contain the number of provided arguments or control commands within the payload. ⌋ (RS_LT_00002, RS_LT_00024)

5.1.1.6 Conditional "ns-Timestamp"

Like specified above (refer PRS_Dlt_01004), the TMSP2 (ns-Timestamp) is added to the Base Header in case the Log and Trace message is a Data (Verbose Mode or Non-Verbose Mode), otherwise the TMSP2 is not part of the Base Header.

The conditional Timestamp is used to add timing information on when a Dlt message has been generated.

[PRS_Dlt_01012] Format of ns-Timestamp ⌈The length for the ns-timestamp shall be 9 byte:

- The lower 4 byte / uint32 shall be the nanoseconds part of the timestamp.
- The upper 5 byte / 40 bits shall be the second's part of the timestamp.

The time shall start from 1970-01-01, 00:00:00,00000, i.e. this timestamp shall be derived from an absolute / global time that has a Synchronized Time Base.⌋ (RS_LT_00017)

Note:
0 to 1.099.511.627.776s ~ 34.841 years
0 to 999999999ns [0x3B9A C9FF];
Invalid value in nanoseconds: [0x3B9A CA00] to [0x3FFF FFFF];
Bit 30 and 31 are reserved in this case.

[PRS_Dlt_01013] Format of ns-Timestamp for ECUs without a synchronized time base ⌈If a specific ECU can't provide an absolute time starting from 1970-01-01, 00:00:00,00000 time, the bit 31 in the nanoseconds field shall be set and the time shall start from the ECU startup.⌋ (RS_LT_00017)

[PRS_Dlt_01014] Substance of the ns-Timestamp ⌈The ns-Timestamp value shall hold the time at the moment an LT User calls the LT module and hands over its LT content.⌋ (RS_LT_00017)

5.1.1.7 Conditional "Message ID"

Like specified above (refer PRS_Dlt_01005), the MSID (Message ID) is added to the Base Header in case the Log and Trace message is a Data Message in Non-Verbose Mode, otherwise the MSID is not part of the Base Header.

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

Note: More details can be found in subchapter *5.1.3.1* Payload in Non-Verbose Mode.

5.1.2 Extension Header

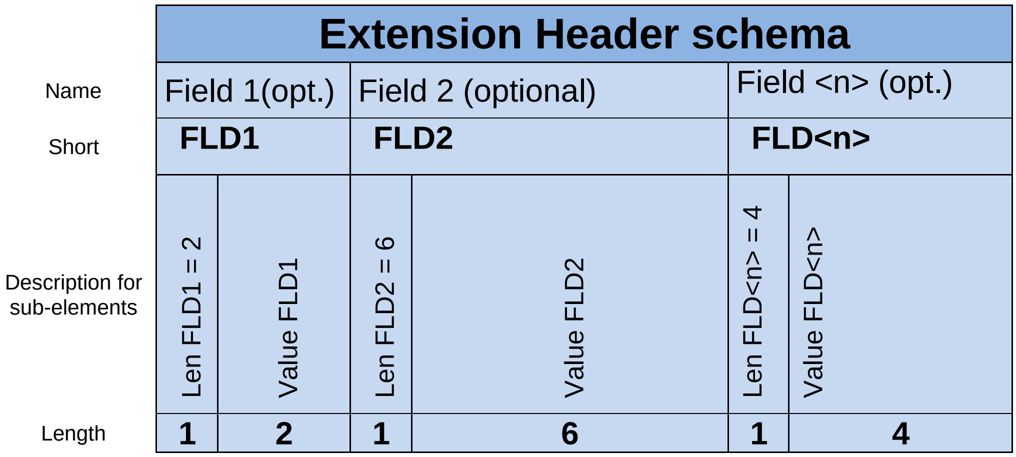
The Extension Header contains additional data that facilitates the interpretation of the pure LT content. Thus, further properties of the LT content, such as the exact origin, are transmitted here.

In case one of the following bits of the "HTYP2"-field in the Base Header are set to ‘1’, additional information is transmitted which are defined in the Extension Header format:

- Bit 2: WEID (With ECU ID)
- Bit 3: WACID (With App- and Context ID)
- Bit 4: WSID (With Session ID)
- Bit 8: WSFLN (With Source File Name and Line Number)
- Bit 9: WTGS (With Tags)
- Bit 10: WPVL (With Privacy Level)
- Bit 11: WSGM (With Segmentation)

The basic design principles for the Extension Header are:

- All of its fields are optional and therefore the complete Extension Header is optional.
- Whether a specific field needs to be added to the Extension Header is indicated by the above mentioned bits ("flags") from the "HTYP2"-field in the Base Header.
- The order of the fields in the Extension Header is defined by the order of the corresponding flags in the "HTYP2"-field from the Base Header.
- A field consist of a length specifier and the value itself (there are a few exceptions to this).



	Extension Header schema					
Name	Field 1(opt.)		Field 2 (optional)		Field <n> (opt.)	
Short	**FLD1**		**FLD2**		**FLD<n>**	
Description for sub-elements	Len FLD1 = 2	Value FLD1	Len FLD2 = 6	Value FLD2	Len FLD<n> = 4	Value FLD<n>
Length	**1**	**2**	**1**	**6**	**1**	**4**

Figure 9 – Schema for the Extension Header

The length information for a specific field can also be '0'. In this case, no field value is provided and the field ends after the length byte.

In order to allow for future expansions of the Extension Header without breaking backward compatibility, all further fields in the future must start with a 1 byte length information. In this way, an implementation according to the current specification can always move from field to field (and thus finally also to the end of the header), even if it can't interpret all of the field values.
Future field elements in the Extension Header are enabled by using the reserved flags in the "HTYP2"-field from the Base Header: currently bits 12 – 31 (marked as "reserved by AUTOSAR for future usage").
As a consequence for all new fields in the future / for all currently "reserved" flags: the number of flags in "HTYP2" which are set to "1" have to be equal with the number of length information added to the Extension Header.

[PRS_Dlt_01015] Locate Extension Header after Base Header ⌈If the Extension Header gets used, it shall be directly attached after the Base Header fields.⌋ (RS_LT_00002)

[PRS_Dlt_01016] Sequence of the fields in the Extension Header ⌈The fields are to be optionally added to the Extension Header depending on and in the sequence of the corresponding flags in the "HTYP2"-field from the Base Header.⌋ (RS_LT_00002)

The Extension Header with all of its currently defined optional fields looks as follow:

Extension Header	ECU	0	8bit length for ECU-ID; (number of bytes)
		1	ASCII characters for ECU-ID;
		2	
		…dyn	
	APID	dyn1	8bit length for Application-ID; (number of bytes)
		dyn1 + 1	ASCII characters for Application-ID;
		dyn1 + 2	
		dyn1 + …	
	CTID	dyn2	8bit length for Context-ID; (number of bytes)
		dyn2 + 1	ASCII characters for Context-ID;
		dyn2 + 2	
		dyn2 + …	
	SEID	dyn3	Session ID 32-bit unsigned integer
		dyn3 + 1	
		dyn3 + 2	
		dyn3 + 3	
	FINA	dyn4	8bit length for the source file name; (number of UTF-8 code units)
		dyn4 + 1	UTF-8 code units for the source file name;
		dyn4 + 2	
		dyn4 + …	
	LINR	dyn5	Line Number 32-bit unsigned integer
		dyn5 + 1	
		dyn5 + 2	
		dyn5 + 3	

	TAGS	dyn6	NOTG	uint8 Number of Tags;						
		dyn6 + 1	Tag_1	8bit length for the name of Tag_1; (number of bytes)						
		dyn6 + 2		ASCII characters for the name of Tag_1;						
		dyn6 + 3								
		dyn6 + …								
		dyn7	Tag_2	8bit length for the name of Tag_2; (number of bytes)						
		dyn7 + 1		ASCII characters for the name of Tag_2;						
		dyn7 + 2								
		dyn7 + …								
		dyn8	Tag_<n>	8bit length for the name of Tag_<n>; (number of bytes)						
		dyn8 + 1		ASCII characters for the name of Tag_<n>;						
		dyn8 + 2								
		dyn8 + …								
	PRLV		Value of PrivacyLevel							
	SGMT	dyn10	8bit length for Segmentation-Info; (number of bytes)							
		dyn10+1	8bit Frame Type :={FirstFrame, ConsecutiveFrame, Last Frame, AbortFrame}							
		dyn10+2	Frame Type Details (<n>bits)	First Frame	64bit Total Length;	Con-secutive Frame	32bit Con-Secutive Frame	LastFrame: 0bit	AbortFrame	8bit Abort
		dyn10+3								-Reason
		dyn10+4								
		dyn10+5								
		dyn10+6								
		dyn10+7								
		dyn10+8								
		dyn10+9								
		dyn11	Length of <future_field>							
		dyn11 + 1	Value of <future_field> ((Template for future extensions))							
		dyn11 + 2								
		dyn11 + …								

Figure 9 - Extended Header

5.1.2.1 Optional ECU-ID

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

[PRS_Dlt_01017] Possibility to send the ECU ID ⌈If the bit 2 (WEID, "With ECU ID") in the "HTYP2"-field of the Base Header is set, the LT-message shall contain the length byte and the string value for the ECU ID, added to the Extension Header.⌋ (RS_LT_00022)

[PRS_Dlt_01018] Length information ⌈The length byte shall be the first byte in the ECU ID field and shall count the number of characters used for the ECU ID.⌋ (RS_LT_00022)

[PRS_Dlt_01019] ECU ID format「The coding of the ECU ID shall contain only ASCII characters without a special terminating item like the NUL-character (\0) at the end.」(RS_LT_00022)

Note: The string end is only given by the Length information for the ECU-ID.

5.1.2.2 Optional Application ID and Context ID

The Application ID is an abbreviation of the application which generates the Dlt message.
The Context ID is a user defined ID to (logically) group Dlt messages generated by an application.

[PRS_Dlt_01020] Possibility to send the Application ID and Context ID「If the bit 3 (WACID, "With App- and Context ID") in the "HTYP2"-field of the Base Header is set, the LT-message shall contain the length bytes and the string values for the Application ID and the Context ID, added to the Extension Header.⌋ (RS_LT_00021)

[PRS_Dlt_01021] Sequence of Application ID and Context ID「If the Application ID and the Context ID are added to the Extension Header, the Application ID field shall be the first and the Context ID field shall be the second.⌋ (RS_LT_00021)

[PRS_Dlt_01022] Length information of Application ID and Context ID ⌈For each of the two fields (Application ID and Context ID) the length byte shall be the first byte in that ID field and shall count the number of characters used for that ID.⌋ (RS_LT_00021)

[PRS_Dlt_01023] Application ID and Context ID format「The coding of the Application ID and Context ID shall contain only ASCII characters without a special terminating item like the NUL-character (\0).」(RS_LT_00021)

Note: The string ends for Application ID and Context ID are only given by the length specification which precedes each.

[PRS_Dlt_01054]「 "#" (U+0023) as prefix of Context IDs shall be reserved for modelled Log and Trace messages standardized by AUTOSAR.⌋ (RS_LT_00058)

[PRS_Dlt_01055] ⌈ Context IDs for Log and Trace messages defined by stack vendors shall have a "+" (U+002B) prefix, followed by the vendor's numerical identifier converted to a string as per PRS_Dlt_01056, followed by a vendor-defined remainder.⌋ (RS_LT_00058)

[PRS_Dlt_01056] ⌈16-bit vendor-IDs are converted to a 2-char ASCII string using Base62 encoding using the string

"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" as digit sequence.⌋ (RS_LT_00058)
Note: The highest Vendor ID that can be encoded with Base62 in two characters without data loss is 3843 (0x0f03). This ID will be encoded as the string "zz".

Example: Using the Vendor ID 0x07bb, the context ID starts with the string "+Vv" with "Vv" being the Base62-encoded string for 0x07bb.

5.1.2.3 Optional Session ID

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

[PRS_Dlt_01024]⌈If the bit 4 (WSID, "With Session ID") in the "HTYP2"-field of the Base Header is set, the LT-message shall contain the Session ID, added to the Extension Header.⌋(RS_LT_00020)

Note: Since the Session ID is defined to be of 32-bit length, this Session ID field in the Extension Header does NOT have an extra length byte in it.

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

5.1.2.4 Optional Source File Name and Source Line Number

To identify the source of log or trace content some information to find the location in the source code shall be added to a Log and Trace message.
Therefore:

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

can be added to the Extension Header.

In a more general way, the source file name is also called "source file identifier": A "source file identifier" constitutes a means to identify the source code file in which a log message originates. That would typically be a filename or filename stem, but could also be a full (or relative) path, or even an entirely different kind, e.g. a hash sum in case filenames are considered to be sensitive data.

[PRS_Dlt_01025] Possibility to send the source file identifier and the source line number⌈If the bit 8 (WSFLN: "With Source File Name and Line Number ") in the "HTYP2"-field of the Base Header is set, the LT-message shall contain the length byte for the source file identifier string and the string value itself and additionally the source line number where the LT message originates from, added to the Extension Header.⌋ (RS_LT_00056)

[PRS_Dlt_01026] Content in the Extension Header for the source file identifier and the source line number ⌈If the source file identifier and the source line number are transmitted in the Extension Header, the following sequence shall be used:

- the length byte for the source file identifier string;
- the string value itself for the source file identifier string;
- the source line number

⌋(RS_LT_00056)

Note: since the source line number is defined to have 32 bit, no additional length byte for the source line number is contained.

[PRS_Dlt_01027] Definition of the length information ⌈The field for the length shall count the number of bytes which the source file identifier consumes. This number also equals the amount of UTF-8 code units.⌋ (RS_LT_00056)

[PRS_Dlt_01028] Source file identifier format ⌈The coding of the source file identifier shall be with UTF-8 code units without BOM and without termination characters.⌋ (RS_LT_00056)

[PRS_Dlt_01029] Substance of the source file identifier ⌈The source file identifier shall contain the indication from where the log or trace content originates.⌋ (RS_LT_00056)

Note: This indication can be made up by the filename stem (filename without an extension) and maybe additionally the filename extension and/or the path (full or partial) to the file can be included.

Alternatively, in case the origin of the log and trace content is considered to be sensitive data, the source file identifier can also be something else, like a hash sum or any other encoded identification.

Note: Up to 255 bytes respectively UTF-8 code units can be used.

[PRS_Dlt_01030] Source Line Number format ⌈The length for the source line number shall be four bytes interpreted as a 32-bit unsigned integer.⌋ (RS_LT_00056)

Note: The Source Line Number starts counting with '1', i.e. the value '0' is not used. Since the length for the line number is statically defined as a 32-bit unsigned integer, no separate length byte shall be added to the Extension Header.

5.1.2.5 Optional Tags

For avoiding bus traffic, especially when logging with Verbose Mode and for tracing, tags could help the application or functional cluster to classify the messages more finely by topic.

[PRS_Dlt_01031] Possibility to send tags for filtering purposes ⌈If the bit 9 (WTGS: "With Tags") in the "HTYP2"-field of the Base Header is set, the LT-message shall contain the following elements in the given sequence:

- the number of attached tags (NOTG);
- for each attached tag:
    - a length byte for the tag name string
    - the string value for the tag name

added to the Extension Header.⌋(RS_LT_00021)

[PRS_Dlt_01032] Definition of the Number of tags ⌈The field "NOTG" (Number of Tags) shall be an 8 bit unsigned integer value and shall count the tags to follow in the Extension Header. Therefore at maximum 255 tags can be added to a LT-message.⌋ (RS_LT_00021)

[PRS_Dlt_01033] Definition of the length information for each tag ⌈The field for the length shall count the number of bytes which the tag name consumes.⌋ (RS_LT_00021)

[PRS_Dlt_01034] Tag name format ⌈The coding of the tag name shall be with ASCII characters without a special terminating item like the NUL-character (\0).⌋ (RS_LT_00021)

Note: The string end is only given by the Length information for the tag name.

5.1.2.6 Optional Privacy Level

The Privacy Level helps to identify the Log and Trace content towards the degree of privacy to it. Logging clients, no matter if in the ECU or outside of the ECU, have the possibility to consider the privacy level at the Log and Trace message to ensure intended and allowed processing of them.

[PRS_Dlt_01035] Possibility to add a privacy level for the containing Log and Trace message ⌈If the bit 10 (WPVL: "With Privacy Level") in the "HTYP2"-field of the Base Header is set, the LT-message shall contain the value for the privacy level of the current LT-message, added to the Extension Header.⌋ (RS_LT_00021)

[PRS_Dlt_01036] Format of the Privacy Level ⌈The length for the Privacy Level shall be one byte unsigned integer.⌋ (RS_LT_00021)

Note: Since the length of the Privacy Level is defined to be one byte, no extra length information is added in the Privacy Level field of the Extension Header.

Note: There is no global definition for the meaning of each single value number of the Privacy Level.

Note: It is up to the external viewer tool or any other instance that interpret or forward the message, to meet this privacy request.

5.1.2.7 Optional Message Segmentation Information

Message Segmentation can be used to transfer a larger amount of payload data that otherwise would have not fit into a single simple LT message. Remember: the total length of a normal, single simple LT message is either limited by the underlying communication protocol / -medium or by the max.value of its "LEN" field in the BaseHeader (16-bit unsigned integer): 65535. In both cases, the available remaining size for the payload is smaller, because the message headers need to be included as well.

[PRS_Dlt_01043] Criteria to use Message Segmentation ⌈ Based on the knowledge of the lower layer frame length limit or the limit of the "LEN" field in the BaseHeader, the L&T module shall decide whether segmentation needs to be used or not.⌋ (RS_LT_00013)

Note: Segmentation should not be used for smaller amounts of payload data, that also fit into a single simple LT message.

[PRS_Dlt_01044] Indication of Message Segmentation ⌈ If Message Segmentation is used, the bit 11 (WSGM, "With Segmentation") in the "HTYP2"-field of the Base Header shall be set and the LT-message shall contain the Segmentation-Information, added to the Extension Header.⌋ (RS_LT_00013)

[PRS_Dlt_01045] Content of the Segmentation-Information in the Extension Header ⌈

The Segmentation-Information shall contain the following elements in the given sequence:

- the length byte for this Segmentation-Information in bytes;
- 8-bit FrameType, which can either be
  - 0 := "FirstFrame";
  - 1 := "ConsecutiveFrame";
  - 2 := "LastFrame";
  - 3 := "AbortFrame";
- x-bit Segmentation details, depending on FrameType:
  - "FirstFrame":
    - 64-bit unsigned integer "TotalLength";
  - "ConsecutiveFrame":
    - 32-bit unsigned integer "SequenceCounter;";

- "LastFrame":
  - 0-bit: n/a; no segmentation details;
- "AbortFrame":
  - 8-bit unsigned integer "AbortReason";
  - 0 - no error/no reason
  - communication time out
  - 2 - insufficient resources
  - 3 - sequence/protocol error

⌋ (RS_LT_00013)

[PRS_Dlt_01046] FrameType sequence for transmission of a Segmented Message ⌈

The segmentation sequence shall be:

- 1 "FirstFrame";
- 0 ... 4.294.967.295 "ConsecutiveFrames": depending on the TotalLength of the segmented data.
- 1 "LastFrame";

⌋ (RS_LT_00013, RS_LT_00018)

Note: "FirstFrame" and "ConsecutiveFrame" use the maximum available size a of regular LT message. The "LastFrame" can be shorter.

[PRS_Dlt_01048] Aborting the sequence ⌈

After the "FirstFrame" but before the "LastFrame", there can be an "AbortFrame" to stop the sequence in case a problem occurred. The already transmitted parts shall be discarded. After an "AbortFrame" was sent, the next allowed FrameType is a "FirstFrame".
⌋ (RS_LT_00013, RS_LT_00018)

[PRS_Dlt_01049] Content of the "TotalLength" information ⌈
The "TotalLength" information shall contain the overall payload data size in bytes that needs to be transmitted in a segmented way. ⌋ (RS_LT_00013)

[PRS_Dlt_01050] Usage of the segmentation "SequenceCounter"⌈

The segmentation SequenceCounter shall only be used in the "ConsecutiveFrame(s)", in case there are any. After each FirstFrame, the SequenceCounter shall start with '0' and can get at maximum '4.294.967.295' in the last "ConsecutiveFrame" before the "LastFrame". There shall be no wrap-around ('4.294.967.295' -> '0', '1' ...). ⌋ (RS_LT_00013, RS_LT_00018)

[PRS_Dlt_01051] Transfer of Payload data blocks ⌈

In case FrameType equals {FirstFrame or ConsecutiveFrame or LastFrame}, the Payload-segment of the LT-messages shall be sequentially filled with data blocks as slices from the overall user-data. The sequence of the data slices must be in line with the transmitted LT-messages:

(with: FirstFrame: FF; ConsecutiveFrame: CF; LastFrame: LF; SequenceCounter: SqCntr)

FF [DataSlice_0], CF_0 [SqCntr = 0; DataSlice1], CF_1 [, SqCntr = 1; DataSlice2], CF_<n> [SqCntr = <n>; DataSlice<n+1>], LF [DataSlice<n+2>]. ⌋ (RS_LT_00013)

5.1.3 Body/Payload format

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

[PRS_Dlt_00314] ⌈If the Extension Header is used, the payload shall adjoin the Extension Header. ⌋ (RS_LT_00013, RS_LT_00023)

[PRS_Dlt_00315] ⌈If the Extension Header is not used, the payload shall adjoin the Base Header. ⌋ (RS_LT_00013, RS_LT_00023)

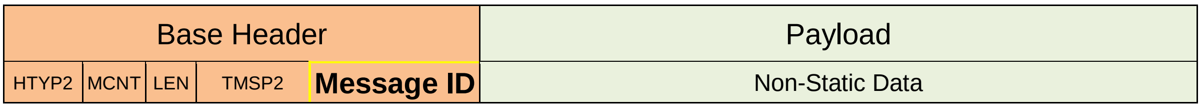
Note: Compare chapter 5.1.2 Extension Header, to see whether the Extension Header is used or not.

5.1.3.1 Payload in Non-Verbose Mode

To be able to transmit parameter values only - without the need of any meta information about them -, without 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 Base Header.

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



Base Header					Payload
HTYP2	MCNT	LEN	TMSP2	**Message ID**	Non-Static Data

Figure 12 – Non-Verbose Mode message

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

[PRS_Dlt_01062] AUTOSAR Message ID range ⌈

Message IDs where bit #31 is set to '1' and bit #30 set to '0' shall be reserved for modelled Log and Trace messages standardized by AUTOSAR.⌋ (RS_LT_00058)

[PRS_Dlt_01053] Vendor-defined Message ID range⌈

Message IDs where bit #31 is set to '1' and bit #30 set to '1' shall be reserved for modelled Log and Trace messages specified by the Framework Provider / Stack Vendor.

Message ID bits #27..#12 shall then hold the vendor's numerical identifier and bits #11..#0 can be used by each vendor for their specific log message identifiers.⌋ (RS_LT_00058)

Note: Assuming the VendorID is 0x0123, a vendor-defined Message ID could be: 0xC012 3A98.

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

- Type Info
- Type Length
- Data Type
- TypeFormat
- TypePrecision
- Variable Info
- Fixed Point

⌋ (RS_LT_00023, RS_LT_00026)

Note: If verbose mode is used instead (see chapter 7.2.5) then 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 message headers:

- Message Type (MSTP)
- Message Info (MSIN)
- Number of arguments (NOAR)

⌋ (RS_LT_00023, RS_LT_00026)

Even if these static data are already specified in that external file (see next chapters for more details) and are therefore not needed as an essential part of the message, it should be allowed in rare cases to send differing static data values in Non-Verbose Mode messages.

[PRS_Dlt_01052] ⌈ In cases where the Message ID is not uniquely related to Context ID and Application ID in the Log and Trace Extract, the fields Context ID and Application ID have to be transmitted separately with Non-Verbose Messages in the Extension Header. This is specifically the case with Standardized logging and tracing.

In case the uniqueness of the MessageID is still given, the Context ID and Application ID shall be recoverable from the external description and a separate transmission is not needed.⌋ (RS_LT_00023, RS_LT_00026)

[PRS_Dlt_01037]⌈Static Data in Non-Verbose Mode messages shall take precedence over the data as specified in the Log and Trace extract file.⌋ (RS_LT_00023, RS_LT_00026)

Note: This case should remain an exception, as otherwise the entire Non-Verbose Mode would become contradictory.

5.1.3.1.1 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 = 295.3 Kelvin

There is a unique Message ID that characterizes 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	295.3	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)

Note: In Verbose Mode the maximum number of arguments can be '255' since the field "NOAR" (NumberOfArguments) is defined to be uint8. In contrast to that, in Non-Verbose Mode the maximum number of arguments is not limited as such by itself. The limit is the overall maximum length of the complete Log and Trace message (headers + payload), which is 65535 bytes because the "LEN" – field is defined as uint16.

5.1.3.1.2 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 see the AUTOSAR_TPS_LogAndTraceExtract [3] for the details.

The software supplier of an application or of the middleware shall provide this description file.

5.1.3.2 Payload in 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.

5.1.3.2.1 Dlt Message Format in General

In Verbose Mode, up to 255 arguments can be transmitted. The information about the payload is provided within the message itself. The payload normally adjoins the Extension Header and consists of one or more arguments. But since the Extension Header is optional, it can be omitted and then the payload adjoins the Base Header. The number of arguments in the payload is specified in the Base 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

- Base Header
- Extension Header (optionally)
- Payload with n Arguments, each consisting of a tuple of Type Info and Data Payload

⌋ (RS_LT_00023, RS_LT_00044)

Base Header	(opt.) Extension 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)

5.1.3.2.2 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.

5.1.3.2.3 Type Info

The Type Info field 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	Type Format (TYFM)
Bit 18 – 23	Type Precision (TYPR)
Bit 24 – 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)	Type Format (TYFM)	Type Precision (TYPR)	*reserved*
Bit	0-3	4	5	6	7	8	9	10	11	12	13	14	15-17	18-23	24-31

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_00782]⌈The bits 15-17 (i.e. Type Format (TYFM)) of the Type Info field define the coding respectively the desired representation format of the later given Data payload. The coding of these three bits depends on the used Types and is restricted to only the following four Types:

- STRG
- SINT
- UINT
- FLOA

⌋(RS_LT_00025, RS_LT_00056)

Note: The Type Format (TYFM) for the Type STRG is identical and fully compatible with the former defined String Coding (SCOD), which has been at the same position, bits 15-17. Compared to the former SCOD, TYFM now extends the usage also to the Types SINT, UINT and FLOA.

[PRS_Dlt_00783]⌈In dependence of the used Type the adjoined Data field is coded respectively shall get interpreted the following way:

used Type	TYFM	meaning
STRG	0x00	ASCII
	0x01	UTF-8
	all other	Reserved and must not be used.
SINT or UINT	0x00	Represent it as base10
	0x01	Represent it as base8
	0x02	Represent it as base16
	0x03	Represent it as base2
	all other	Reserved and must not be used.
FLOA	0x00	implementation-defined
	0x01	Represent it as decimal floating point, similar to printf "%f"
	0x02	Represent it in scientific notation (mantissa, exponent), similar to printf "%e"
	0x03	Represent it as hexadecimal floating point, similar to printf "%a"
	0x04	Represent it in the shortest way, also known as the "general format": either decimal floating point or scientific notation, similar to printf "%g"
	all other	Reserved and must not be used.

⌋(RS_LT_00025, RS_LT_00056)

Note: The mentioned hint "similar to printf" is intended to refer to the C-function "printf()" and its conversion specifiers 'f', 'e', 'a' and 'g' as defined by the C Standard Library <stdio.h>.

Note: The FLOA TYFM '0x04' (= "general format" for floating-point numbers) is intended to be similar to what the conversion specifier 'g' for the C-function "printf()" does. The detailed description to conversion specifier 'g' is more complex than simply "use the shortest out of %e or %f", like written in the table above. For the exact details refer to the C11 standard.

[PRS_Dlt_00784]⌈ The bits 18-23 (i.e. Type Precision (TYPR)) of the Type Info field define the desired precision of the later given Data payload. The coding of these six bits depends on the used Type and is restricted to only the following three Types:

- SINT
- UINT
- FLOA

⌋(RS_LT_00056)

[PRS_Dlt_00785]⌈ In dependence of the used Type the adjoined Data field shall get a precision in the following way:

used Type	TYPR	meaning
SINT or UINT	0	use needed number of digits for the value to be written (similar to printf "%d")
	1..63	minimum number of digits to appear +1 (e.g. TYPR = 3 equals printf "%.4d")
		Larger minimum numbers of digits to appear can't be specified. (e.g. like potentially needed for 128 bit integers in BIN-format)
FLOA	0	use implementation-defined precision;
	1..62	number of digits for precision -1 (e.g. TYPR = 3 equals printf "%.2f");
	63	use precision necessary for loss-less printing of the type, e.g. "%.17e" for Float64 (only for TYFM equals '2', '3' or '4')
		Larger numbers of digits for precision (e.g. like potentially needed for very small float numbers printed as decimal floating point (printf "%.*f")) can't be specified.

⌋(RS_LT_00056)

The table below shows a simplified assembly of Type Info

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

5.1.3.2.3.1 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 Length contains

- 0 for strings (STRG), structs (STRU), raw data (RAWD) and Trace Info (TRAI)
- 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)

5.1.3.2.3.2 Bit Variable Info (VARI)

If Variable Info (VARI) is set, the name and the unit of a variable can be added at the beginning of the Data payload field. Both 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 field. The unit information is to add only in some data types.

Independent from the data type, the name or the unit can be omitted (if not needed) by setting the corresponding length information field to 0.

[PRS_Dlt_00410] ⌈The coding of all text in Variable Info (VARI) shall be with 8-bit characters where each character is within the valid range of the ASCII character set.⌋(RS_LT_00025)

[PRS_Dlt_01038] ⌈The strings in VARI shall be without a special terminating item like the NUL-character (\0).⌋ (RS_LT_00025)

[PRS_Dlt_01039] ⌈If the length information field of the name or the unit is set to 0, the corresponding text field shall be omitted.⌋ (RS_LT_00025)

5.1.3.2.3.3 Bit Fixed Point (FIXP)

If fixed point values are used (SINT or UNIT) for transmission at protocol level but the value should finally represent a floating point number, the Fixed Point (FIXP) bit shall be set. Then 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. If the Fixed Point (FIXP) bit is set, the quantization and the offset are added at the beginning of the Data payload field.

[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)

5.1.3.2.3.4 Bits Type Format (TYFM)

Type Format specifies only the coding of the data field in case it is of Type String (STRG), Type Signed (SINT), Type Unsigned (UINT) and Type Float (FLOA). All other protocol elements keep their default format, like

- strings for parameter name and unit and description are coded with 8-bit characters where each character is within valid range of ASCII character set; or
- integers and floats for length information or fixed point conversion are out of scope for a representation format information.

[PRS_Dlt_00786] ⌈Type Format is a bit field of 3 bit.⌋ (RS_LT_00025, RS_LT_00056)

[PRS_Dlt_00787] ⌈In case the used Type is String (STRG) the following values for Type Format shall be used to encode and decode respectively interpret the adjoined Data field:

- 0x00: ASCII (8-bit characters where each character is within valid range of ASCII character set)
- 0x01: UTF-8
- 0x02 - 0x07: reserved for future use

⌋(RS_LT_00025, RS_LT_00056)

Note: For this case, the TypeFormat (TYFM) is a compatible replacement for the former String Coding (SCOD) bits.

[PRS_Dlt_00788] ⌈In case the used Type is Signed (SINT) or Unsigned (UINT) the following values for Type Format shall be used to request and interpret a requested display representation of the adjoined Data field:

- 0x00: write it as base10
- 0x01: write it as base8
- 0x02: write it as base16
- 0x03: write it as base2
- 0x04 – 0x07: reserved for future use

⌋(RS_LT_00056)

[PRS_Dlt_00789]⌈In case the used Type is Float (FLOA) the following values for Type Format shall be used to request and interpret a requested display representation of the adjoined Data field:

- 0x00: implementation-defined; (used for backward compatibility reasons with the former String Coding (SCOD).)
- 0x01: similar to printf "%f" => display the value in the format "[-]ddd.ddd", i.e. as decimal floating point; e.g.: '123.45';
- 0x02: similar to printf "%e" => display the value in the format "[-]d.ddd**e**±dd", i.e. as scientific notation with mantissa and exponent; e.g.: '1.2345e+2';
- 0x03: similar to printf "%a" => display the value in the format "[-]**0x**h.hhhh**p**±d", i.e. as hexadecimal floating point; e.g.: '0x1.edccccp+6';
- 0x04: similar to printf "%g" => Use the shortest representation: %e or %f, also known as "general format" for floating-point numbers.
- 0x05 – 0x07: reserved for future use

⌋(RS_LT_00056)

[PRS_Dlt_00790]⌈Type Format shall be set and used for interpretation if Type String (STRG), Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) is set.⌋ (RS_LT_00025, RS_LT_00056)

[PRS_Dlt_00791]⌈If Trace Info (TRAI) is set, Type Format shall be set and used for interpretation of the trace data string like for Type String (STRG).⌋ (RS_LT_00044, RS_LT_00025, RS_LT_00056)

[PRS_Dlt_00792]⌈If Type Array (ARAY) is set in combination with Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA), Type Format shall be set and used for interpretation.⌋ (RS_LT_00056)

[PRS_Dlt_00793]⌈If Type Struct (STRU) is set, Type Format shall be set and used for interpretation in each single substructured Type Info field in case the addressed Data field is of Type String (STRG), Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA).⌋ (RS_LT_00025, RS_LT_00056)

[PRS_Dlt_00794]⌈For Data field types

- other than Type String (STRG), Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) or
- other than Arrays of Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) or

- other than Type String (STRG), Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) as sub-elements in Struct

the Type Format should be filled with '0' and shall be ignored.⌋ (RS_LT_00044, RS_LT_00025, RS_LT_00056)

5.1.3.2.3.5 Bits Type Precision (TYPR)

Type Precision applies only for Data of Type Signed (SINT), Type Unsigned (UINT) and Type Float (FLOA).

[PRS_Dlt_00795] ⌈Type Precision (TYPR) is a bit field of 6 bit.⌋ (RS_LT_00056)

[PRS_Dlt_00796] ⌈In case the used Type is Signed (SINT) or Type Unsigned (UINT) the following values for Type Precision (TYPR) shall be used to request and interpret a requested minimum number of digits to appear in the requested TYFM number base (e.g. like "base2" or "base16") for the adjoined Data field:

- 0: use needed number of digits for the value to be printed (similar to printf "%d"); at least one digit shall be printed; in that way, also the character '0' is written and the digit stays not empty. Padding is not used.
- 1 - 63: minimum number of digits to appear +1 (e.g. TYPR = 3 equals printf "%.4d");
  If the converted value requires more digits, the TYPR is ignored and the complete number is written.

  If the converted value requires fewer digits, the value is padded on the left.

⌋(RS_LT_00056)

Note: For Type Format equaling to "base16", "base8" or "base2" a padding with '0' may be used, otherwise ("base10") a space padding may be used in case the minimum number of digits is longer than the value to be written.

[PRS_Dlt_00797] ⌈In case the used Type is Float (FLOA) and TYFM equals '0' ("implementation-defined") or '1' ("decimal floating point" / printf %f), the following values for Type Precision (TYPR) shall be used to request and interpret a requested number of digits to appear after the radix point for the adjoined Data field:

- 0: use implementation-defined number of digits to appear after the radix character;
- 1 - 63: number of digits to appear after the radix character -1 (e.g. TYPR = 3 equals printf "%.2f")

⌋(RS_LT_00056)

[PRS_Dlt_00798] ⌈In case the used Type is Float (FLOA) and TYFM equals '2' ("scientific notation" / printf %e) or '3' ("hexadecimal floating point " / printf %a), the following values for Type Precision (TYPR) shall be used to request and interpret a requested number of digits to appear after the radix point for the adjoined Data field:

- 0: use implementation-defined precision;
- 1 - 62: number of digits to appear after the radix character -1 (e.g. TYPR = 3 equals printf "%.2e")
- 63: use precision necessary for loss-less printing of the type, e.g. "%.17e" for Float64

⌋(RS_LT_00056)

Note: If TYPR equals '63' and therefore a "loss-less printing" of the float value is requested: depending on the type length of the concerned float-value, the needed precision differs:

Type Length (TYLE)	Length	Number of needed decimal digits for loss-less printing:
2	16 bit	5
3	32 bit	9
4	64 bit	17
5	128 bit	36

Due to the nature of "decimal floating point" (compare **[PRS_Dlt_00797]**), the "loss-less printing" option is not foreseen for TYFM equals '0' or '1'.

[PRS_Dlt_00799] ⌈In case the used Type is Float (FLOA) and TYFM equals '4' ("general format" for floats) the following values for Type Precision (TYPR) shall be used to request and interpret a requested number of significant digits for the adjoined Data field:

- 0: use implementation-defined precision;
- 1 - 62: number of significant digits (e.g. TYPR = 3 equals printf "%.3g")
- 63: use precision necessary for loss-less printing of the type, e.g. "%.17g" for Float64

⌋(RS_LT_00056)

[PRS_Dlt_00800] ⌈Type Precision shall be set and used for interpretation if Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) is set.⌋ (RS_LT_00056)

[PRS_Dlt_00801]「Type Precision shall be set and used for interpretation if Type Array (ARAY) is set in combination with Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA).」(RS_LT_00056)

[PRS_Dlt_00802]「If Type Struct (STRU) is set, Type Precision shall be set and used for interpretation in each single sub-structured Type Info field in case the addressed Data field is of Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA).」(RS_LT_00056)

[PRS_Dlt_00803]「For Data field types

- other than Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) or
- other than Arrays of Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) or
- other than Type Signed (SINT), Type Unsigned (UINT) or Type Float (FLOA) as sub-elements in Struct

the Type Precision should be filled with '0' and shall be ignored.」(RS_LT_00044, RS_LT_00025, RS_LT_00056)

5.1.3.2.3.6 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. In all other cases 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 and the Name 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	Unsigned 16-bit integer
x	Name	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)

5.1.3.2.3.7 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	Unsigned 16-bit integer
16	Length of Unit	Unsigned 16-bit integer
x	Name	String (name of variable)
x	Unit	String (unit of variable)
If **Fixed Point (FIXP)** is set in Type Info		
32	Quantization	32-bit float in binary representation according to IEEE 754-2008
32 / 64 / 128	Offset	Signed integer - with the length of at least 32 bit. The length shall be: 32 bit if Type Length (TYLE) equals 1,2 or 3 64 bit if Type Length (TYLE) equals 4 or 128 bit if Type Length (TYLE) equals 5
8/16/32 / 64 / 128	**Data**	Length depends on TYLE

Table 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 value in binary representation according to IEEE 754-2008. ⌋ (RS_LT_00044)

[PRS_Dlt_00387] ⌈The Offset field is a signed integer field with at least 32 bit. If the TYLE equals 4 the Offset field shall be a 64 signed integer field and if the TYLE equals 5 the Offset field shall be a 128 signed 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)

5.1.3.2.3.8 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 value in binary representation according to IEEE 754-2008.

Variable Info (VARI) is optional.

Length in bit	Name	Description
If **Variable Info (VARI)** is set in Type Info		
16	Length of name	Unsigned 16-bit integer
16	Length of unit	Unsigned 16-bit integer
x	Name	String (name of variable)
x	Unit	String (unit of variable)
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 754:2008:

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

Table 5-5 Definition of Type Length according to IEEE 754:2008

⌋ (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)

5.1.3.2.3.9 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_00156] ⌈At the beginning of the Data Payload, a 16-bit unsigned integer specifies the length of the string (provided in the Data field) in byte. ⌋ (RS_LT_00025)

Note: The string end is only defined by this length information.

[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	Unsigned 16-bit integer
If **Variable Info (VARI)** is set in Type Info			
	16	Length of name	Unsigned 16-bit integer
	x	Name	String (name of variable)
x		**Data** string	Data string without a special terminating item like the NUL-character (\0)

Table 5-6 Data Payload of Type String (STRG)

⌋ (RS_LT_00025)

Note: The Data string end is only given by the "Length of string" information.

5.1.3.2.3.10 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	Unsigned 16-bit integer
	16	Length of Unit	Unsigned 16-bit integer
	x	Name	String (name of current dimension)
	x	Unit	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)

5.1.3.2.3.11 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	Unsigned 16-bit integer
x	Name	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)

5.1.3.2.3.12 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 with 8-bit characters where each character is within valid range of ASCII character set.⌋ (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	Unsigned 16-bit integer
	x	Name	String (description of variable)
x		**Data**	Raw data

Table 5-9 Data Payload of Type Raw (RAWD)

⌋ (RS_LT_00044)

5.1.3.2.3.13 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. ⌋ (RS_LT_00044)

[PRS_Dlt_00173] ⌈The trace data string without a special terminating item like the NUL-character (\0) shall follow. ⌋ (RS_LT_00044)

Note: Type Format (TYFM) specifies the coding of the trace data string

[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 (in byte)	Unsigned 16-bit integer
x	**Trace Data String**	String (like name of module / function in packet)

Table 5-10 Data Payload of Trace Info (TRAI)

⌋ (RS_LT_00044)

5.1.3.2.4 Example of representation of natural data type argument

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

The Type Info is a 32-bit field that describes the Data. In this example, it defines the variable type (unsigned integer), its length (8 bit) and the presence of Variable Info (VARI) that describes the name and unit of the variable.

Variable Info is following with two 16-bit unsigned integers describing the length of the Name and the Unit of the variable.
Two 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	11	Unsigned 16-bit integer
16	Length of unit	7	Unsigned 16-bit integer
88 (11*8)	Name	temperature	String (name of variable)
56 (7*8)	Unit	Celsius	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 field bit combinations
The following table shows all combinations of valid settings in Type Info sorted according to the bit position in Type Info. Consider:

- x – mandatory for this type,
- $x_{(1)}$ – mandatory in case array consists of type for which TYFM respectively TYPR is mandatory,
- $x_{(2)}$ – mandatory in TypeInfo for struct sub-elements in case those consists of types for which TYFM respectively TYPR is mandatory,
- (x) – mandatory: an ARAY consists of elements from one of that types;
- – optional,
- empty – (not allowed for this type)

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 TYFM			18-23 TYPR						24-31 RESERVED	
x	x	x	x	x							o														
x	x	x	x		x						o	o			x	x	x	x	x	x	x	x	x		
x	x	x	x			x					o	o			x	x	x	x	x	x	x	x	x		
x	x	x	x				x				o				x	x	x	x	x	x	x	x	x		
									x		o				x	x	x								
										x	o														
													x		x	x	x								
x	x	x	x	(x)	(x)	(x)	(x)	x			o	o			$x_{(1)}$	$x_{(1)}$	$x_{(1)}$	$x_{(1)}$	$x_{(1)}$	$x_{(1)}$	$x_{(1)}$	$x_{(1)}$	$x_{(1)}$		
											o			x	$x_{(2)}$	$x_{(2)}$	$x_{(2)}$	$x_{(2)}$	$x_{(2)}$	$x_{(2)}$	$x_{(2)}$	$x_{(2)}$	$x_{(2)}$		
											o														

Table 5-12 Assembly of valid settings in Type Info

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)	Type Format (TYFM)	Type Precision (TYPR)
Type Bool (BOOL)	x	o			
Type Signed Integer (SINT)	x	o	o	x	x
Type Unsigned Integer (UINT)	x	o	o	x	x
Type Float (FLOA)	x	o		x	x
Type Array (ARAY)	x	o			$x_{(1)}$
Type String (STRG)		o		x	
Type Raw (RAWD)		o			
Trace Info (TRAI)				x	
Type Struct (STRU)		o			$x_{(2)}$

Table 5-13 Assembly of valid settings in Type Info (o – optional, x – mandatory for this type, x(1) & x(2): see Table 5 12; empty – not allowed for this type)

Using the Verbose Mode helps to understand, analyze and debug the application.

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 level, 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	ResetToFactoryDefault	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

⌋ (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 Base Header, an optional Extension Header, and a payload. The payload consists of one or more tuples of the Service ID, transmitted as 32-bit unsigned integer and the contained parameters. ⌋ (RS_LT_00032)

Base Header					**Payload**			
					Command 1		opt.: Command <2…n>	
HTYP2	MCNT	LEN	MSIN	**NOAR**	**Service ID**	Parameters	**Service ID**	Parameters

Figure 15 –Verbose Mode message

Note: For most of the Commands, the Extension Header is not needed (Fig. 15). It is needed e.g. for the "Call SWC Injection" command.

[PRS_Dlt_01040] Usage of NOAR in Control Messages ⌈If a Control Message is sent, the Base Header field "NOAR" (Number of arguments) shall contain the number of Service IDs / Commands sent in that Control Message.⌋ (RS_LT_00032)

[PRS_Dlt_01041] Avoid overlapping requests or responses for a single Service ID ⌈If there exists a specified response to a certain request / Service ID, each request shall be responded before the next request for the same Service ID is allowed to be sent out.⌋ (RS_LT_00032)

[PRS_Dlt_01042] Order of request execution and response ⌈If a Control Message is sent with more than one Service IDs / Commands ("NOAR" > 1), the requests shall be executed in the same sequence as in the Control Message. The response for one request shall be generated, before the next request is processed. The responses to the commands shall be sent in the same sequence as in the request Control Message.⌋ (RS_LT_00032)

Note: The generated responses can be sent all in one Control Message again ("NOAR" > 1) or can be split into more Control Messages.

If one Control Message contains for example

- 1) Request "GetLevel_x";
- 2) Request "SetLevel_x";
- 3) Request "GetLevel_x";

the result in the response needs to be:

- 1) Response "GetLevel_x = old level";
- 2) Response "SetLevel_x = OK for new level ";
- 3) Response "GetLevel_x = new level ";

5.3.1 Set Log Level

[PRS_Dlt_01057] ⌈

Service name:	SetLogLevelLong		
Service_ID [hex]	`0x000000025`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	**applIdLength**	`uint8`	+One byte length info: number of characters for Application-ID; If this field equals 0, the command shall be ignored. +
2	**applicationId**	`uint8`	+Representation of the Application ID. +
3	**contextIdLength**	`uint8`	+One byte length info: number of characters for Context-ID; If this field equals equals 0, the command shall be ignored.+
4	**contextId**	`uint8`	+Representation of the Context ID. +
5	**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
6	***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_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)

[PRS_Dlt_01058] ⌈

Service name:	SetTraceStatusLong		
Service ID [hex]	`0x0000026`		
:			
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	**applIdLength**	`uint8`	+One byte length info: number of characters for Application-ID; If this field equals 0, the command shall be ignored.+
2	**applicationId**	`uint8`	+Representation of the Application ID. +

3	**contextIdLength**	`uint8`	+One byte length info: number of characters for Context-ID; If this field equals 0, the command shall be ignored.+
4	**contextId**	`uint8`	+Representation of the Context ID. +
5	**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
6	**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:	Set the pass though range for log messages for a given combination 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) ' contextDescLen (uint8) ' contextDesc[] (uint8[]) ' appDescLen (uint8) ' 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)

[PRS_Dlt_01059] ⌈

Service name:	GetLogInfoLong		
Service ID [hex]	0x00000027		
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	**applicationIdLength**	`uint8`	+One byte length info: number of characters for Application-ID; If this field equals equals 0, the command shall be ignored.+
3	**applicationId**	`uint8`	+Representation of the Application ID. +
4	**contextIdLength**	`uint8`	+One byte length info: number of characters for Context-ID; If this field equals 0, the command shall be ignored.+
5	**contextId**	`uint8`	+Representation of the Context ID. +
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[])
			  appIdLen(uint8)
			  appID (uint8[4])
			  contextIdCount (uint16)
			  contextIdInfoList[] (struct[])
			    contextIdLen(uint8)
			    contextId (uint8[4])
			    logLevel (enum 0x00 .. 0x06)
			    traceStatus (uint8)
			    contextDescLen (uint8)
			    contextDesc[] (uint8[])
			  appDescLen (uint8)
			  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)

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)

[PRS_Dlt_01060] ⌈

Service name:	GetTraceStatusLong		
Service ID [hex]:	`0x00000028`		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***applicationIdLength***	`uint8`	One byte length info: number of characters for Application-ID;
2	***applicationId***	`uint8`	Addressed Application ID
3	***contextIdLength***	`uint8`	One byte length info: number of characters for Context-ID;
4	***contextId***	`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)

[PRS_Dlt_01061] ⌈

Service name:	SetLogChannelAssignmentLong		
Service ID [hex]:	0x00000029		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Request Parameter			
Number	***Name***	**Type**	**Description**
1	***applicationIdLength***	uint8	One byte length info: number of characters for Application-ID;
2	***applicationId***	uint8	Addressed Application ID
3	***contextIdLength***	uint8	One byte length info: number of characters for Context-ID;
4	***contextId***	uint8	Addressed Context ID
5	***logChannelName***	uint8	Name of the addressed LogChannel
6	***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 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. ⌋ (RS_LT_00032)

[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. ⌋ (RS_LT_00032)

[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. ⌋ (RS_LT_00032)

[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.		

⌋ (RS_LT_00032)

5.3.19 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
- 0x24 SyncTimeStamp

⌋ (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 the ECU ID. Therefore, it is important to store some additional information by the receiving external client.

For additionally storing useful information like the timestamp of the receiver and the ECU ID a Storage Header shall be added in front of every received Dlt message. Because the ECU ID can be omitted by the sending Dlt side, the receiver shall add this information at receiving time. The Timestamp is also for a better calculation 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”+0x02 in Hex 0x 44 4C 54 02
4	9		Timestamp	
		5	seconds	Unsigned integer 32 bit seconds since 01.01.1970 (Unix time)
		4	nanoseconds	Singed integer 32 bit nanoseconds of the second (between 0 – 999.999.999)
12	1 + n		ECU ID	
		1	Length of ECU ID	1 byte length info: number of characters for ECU-ID;
		n	ECU ID	<n> ASCII characters for ECU-ID
13 + n	**Dlt log or trace message**			
			Base Header	
			Extension Header	(if contained in the message)
			Payload	

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

⌋ (RS_LT_00002)

Note: The Storage Header is applied to Data Messages (Verbose Mode and Non-Verbose Mode) and to Control Messages.

[PRS_Dlt_00427] ⌈The first entry in the Storage Header shall be a pattern 0x 44 4C 54 02 (“DLT”+0x2). ⌋ (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)

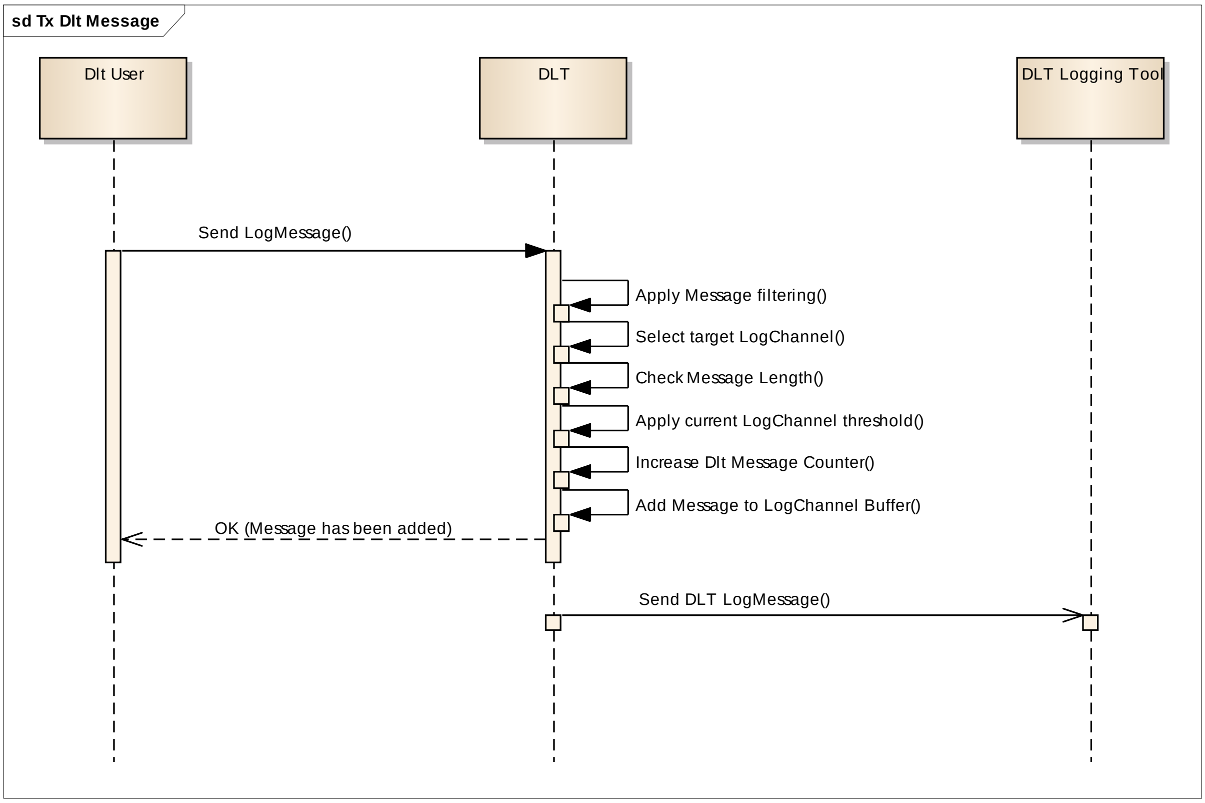
5.5 Sequences (lower layer)

5.5.1 States

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

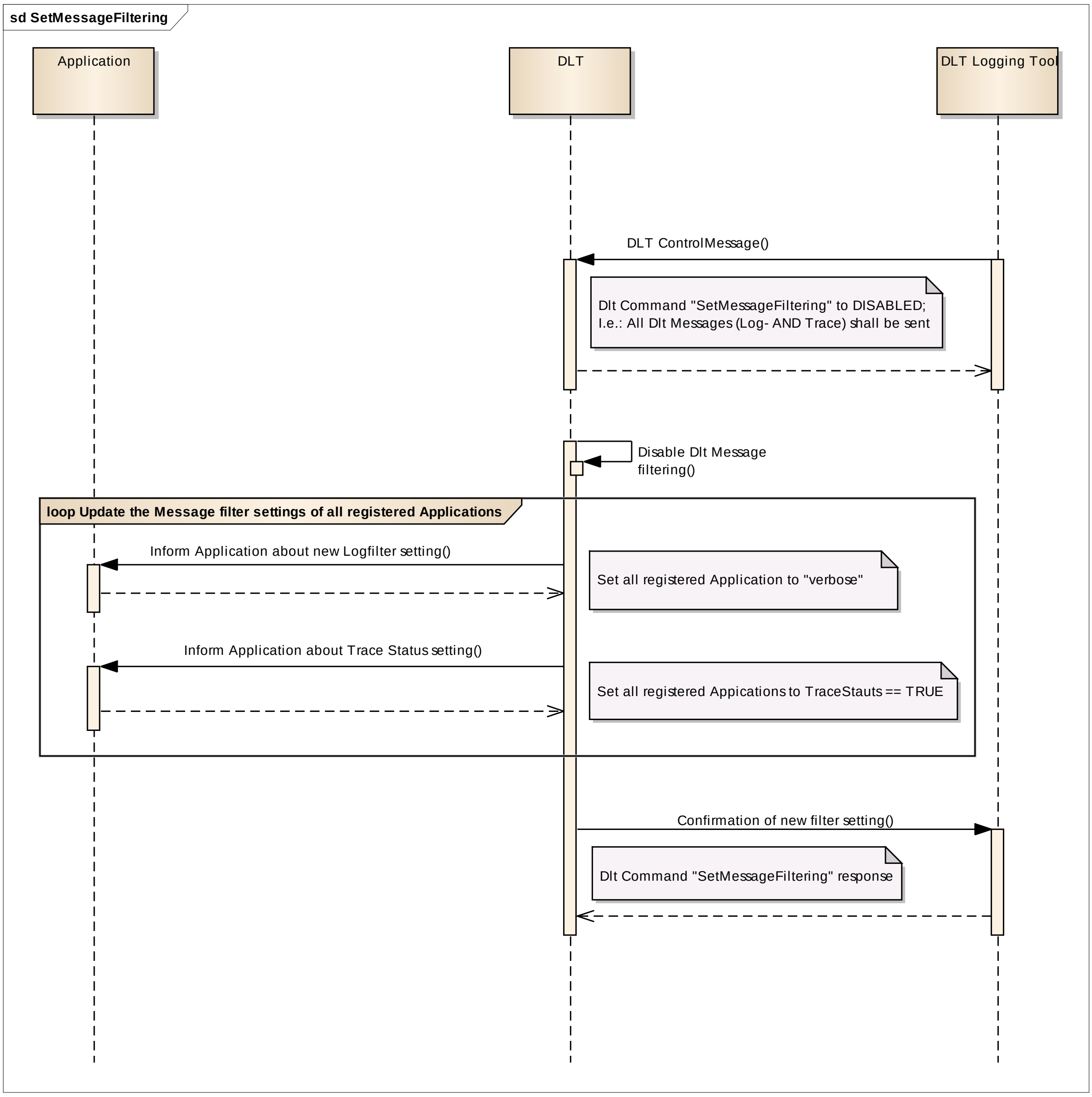
5.5.2 Control flow / Transitions

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()

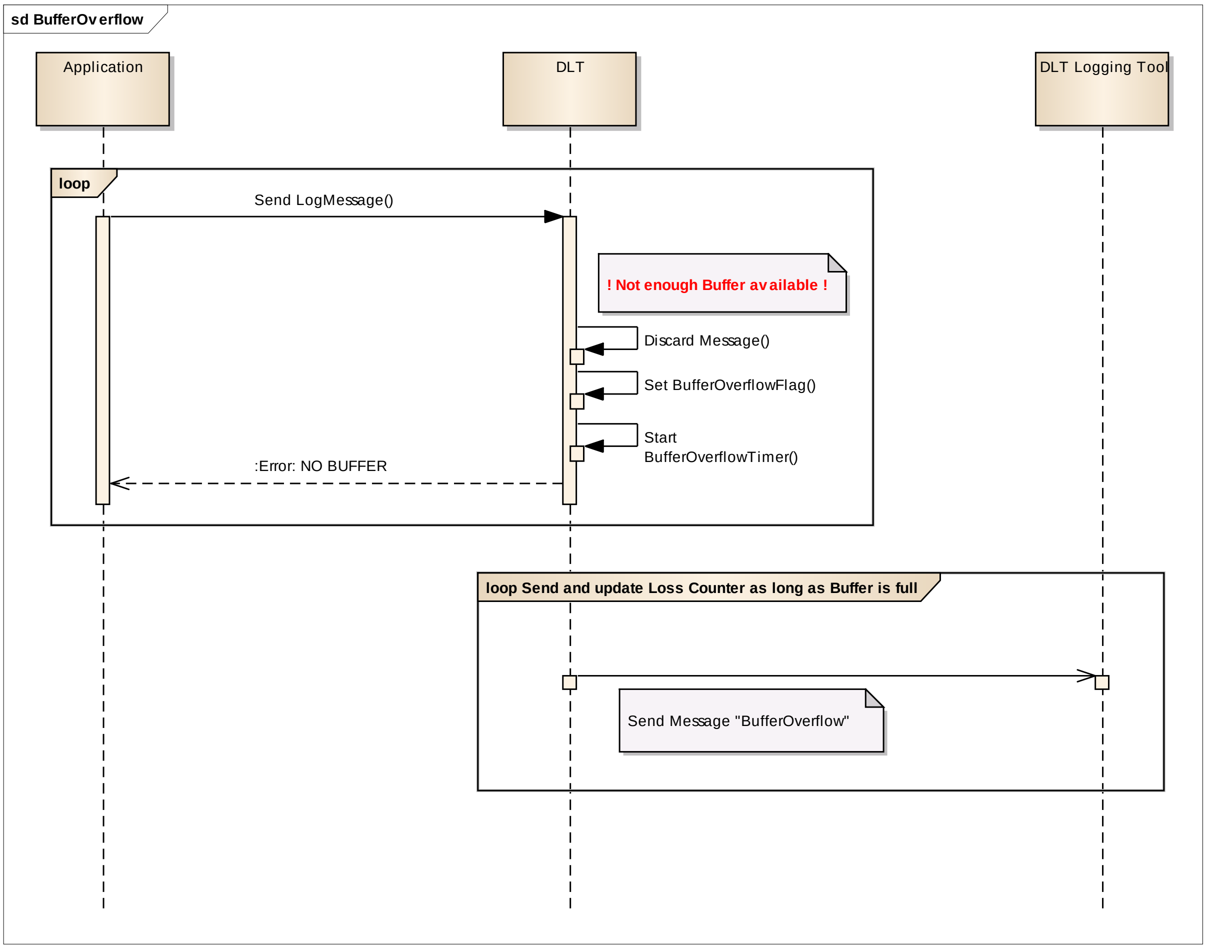
Sequence 1 – Transmission of Dlt Data Message

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.⌋ (RS_LT_00037)

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. ⌋ (RS_LT_00037)

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

⌋ (RS_LT_00032)

[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". ⌋ (RS_LT_00032)

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

⌋ (RS_LT_00032)

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. ⌋ (RS_LT_00030)

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 *5.1.1.4* Conditional "Message Info" 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

7 Configuration specification

This chapter lists all parameter the Dlt Protocol uses.

7.1 Base Header

Long Name	Short Name	Description
Header Type 2	HTYP2	Meta information about the Headers
Message Counter	MCNT	Message counter of Dlt messages
Length	LEN	Length of Dlt Message
Message Info (c)	MSIN	Meta information about the payload
Number of Arguments (c)	NOAR	Number of arguments contained in payload
ns-Timestamp (c)	TMSP2	Timestamp with nanoseconds resolution
Message ID (c)	MSID	Unique message ID for

7.1.1 Header Type 2 (HTYP2)

Long Name	Short Name	Description
Content Information	CNTI	Verb/Non-Verb Data msg or Control msg.
With ECU ID	WEID	Flag for ECU ID field usage
With App- and Context ID	WACID	Flag for App- and Context ID fields usage
With Session ID	WSID	Flag for Session ID field usage
Version number	VERS	Contains the used Dlt Protocol Version
With Source File Name and Line Number	WSFLN	Flag for Source File Name and Line Number fields usage
With Tags	WTGS	Flag for tags field usage
With Privacy Level	WPVL	Flag for Privacy Level field usage

7.1.2 Message Info (MSIN)

Long Name	Short Name	Description
Message Type	MSTP	Identification of the type of Dlt message
Message Type Info	MTIN	Additional information of the message type

7.2 Extension Header

Long Name	Short Name	Description
ECU ID (optional)	ECU	Name of ECU
Application ID (optional)	APID	Application ID
Context ID (optional)	CTID	Context ID
Session ID (optional)	SEID	Session ID
File Name	FINA	LT- message origin: Source file name
Line Number	LINR	LT- message origin: Line number in file
Tags	TAGS	Tags for filtering purposes
Privacy Level	PRLV	Privacy level of message content

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.