VECTOR

MICROSAR Diagnostic Log and Trace on Xcp

Technical Reference

Diagnostic Log and Trace

Version 2.1.1

Authors	Patrick Markl, Oliver Reineke, David Zentner
Status	Released

Document Information

History

Author	Date	Version	Remarks
Patrick Markl	2011-06-20	1.0.0	Creation
Patrick Markl	2013-03-26	1.0.1	ESCAN00065965: Extended integration directions.
Oliver Reineke	2013-07-11	1.1.0	ESCAN00068275: AR4-292: Reporting of DET and DEM errors via DLT
Klaus Emmert	2013-10-07	1.1.1	Typos and formats
David Zentner	2015-03-24	1.2.0	Feature: DLT with AUTOSAR functionality and DLT Transport Layer
David Zentner	2015-11-20	2.0.0	ESCAN00086655: Create two different Technical References for Dlt_OnXcp and Dlt_Autosar.
David Zentner	2016-03-07	2.1.0	Security Mechanism added.
David Zentner	2017-07-03	2.1.1	ESCAN00092156

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DiagnosticLogAndTrace.pdf	V1.2.0
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	V2.2.1V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0
[4]	Vector	UserManual_AMD.pdf	V1.2.1
[5]	Vector	ASAP2 Tool-Set User Manual	V4.3

Scope of the Document**:**

This technical reference describes the use of the DLT module for tracing and logging purposes.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

1 Component History 7

2 Introduction 8
2.1 Architecture Overview 9

3 Functional Description 10
3.1 Features 10
3.1.1 Supported features 10
3.1.2 Deviations 10
3.2 Initialization 11
3.3 States 11
3.4 Main Functions 11
3.5 Error Handling 11
3.5.1 Development Error Reporting 11
3.5.1.1 Parameter Checking 12
3.5.2 Production Code Error Reporting 13

4 Integration 14
4.1 Scope of Delivery 14
4.1.1 Static Files 14
4.1.2 Dynamic Files 15
4.2 Include Structure 16
4.3 Compiler Abstraction and Memory Mapping 16
4.4 Critical Sections 17
4.5 Common Configuration Steps 17
4.6 DLT on XCP 18
4.6.1 XCP Event 18
4.6.2 XCP: Logging of verbose and non-verbose DLT messages 19
4.6.3 Workflow McData 20
4.6.4 DLT Reporting with CANoe 22
4.6.5 DLT Reporting with CANape 24

5 API Description 27
5.1 Type Definitions 27
5.2 Services provided by DLT 28
5.2.1 Dlt_InitMemory 28
5.2.2 Dlt_Init 28
5.2.3 Dlt_MainFunction 29
5.2.4 Dlt_GetVersionInfo 29

5.2.5 Dlt_DetForwardErrorTrace 30
5.2.6 Dlt_DemTriggerOnEventStatus 30
5.2.7 Dlt_SendLogMessage 31
5.2.8 Dlt_SetState 32
5.2.9 Dlt_GetState 33
5.3 Services used by DLT 33

6 Glossary and Abbreviations 35
6.1 Glossary 35
6.2 Abbreviations 35

7 Contact 36

Illustrations

Figure 2-1 AUTOSAR 4.x Architecture Overview 9
Figure 4-1 Include structure DLT 16
Figure 4-2 Configuration of non-verbose messages 19
Figure 4-3 Open XCP/CCP option 22
Figure 4-4 Selection of the XCP measurement object for DLT logging of DET reports 23
Figure 4-5 Enable the symbolic option by clicking the button [sym] in CANoe's toolbar 24
Figure 4-6 CANape's symbol explorer for selection of the DLT variable 25
Figure 4-7 Select the DLT variable for measurement in a text window 25
Figure 4-8 Logging of DLT messages using CANape's text window. 26

Tables

Table 1-1 Component history 7
Table 3-1 Supported AUTOSAR standard conform features 10
Table 3-2 Deviations from AUTOSAR standard 11
Table 3-3 Service IDs 12
Table 3-4 Errors reported to DET 12
Table 3-5 Development Error Reporting: Assignment of checks to services 13
Table 4-1 Static files 14
Table 4-2 Available configuration options of the DLT module 15
Table 4-3 Generated files 15
Table 4-4 Compiler abstraction and memory mapping 17
Table 4-5 Exclusive Areas of the DLT module 17
Table 4-6 DLT functions and their corresponding XCP events 18
Table 5-1 Type definitions 28
Table 5-2 Dlt_InitMemory 28
Table 5-3 Dlt_Init 29
Table 5-4 Dlt_MainFunction 29
Table 5-5 Dlt_GetVersionInfo 30
Table 5-6 Dlt_DetForwardErrorTrace 30
Table 5-7 Dlt_DemTriggerOnEventStatus 31
Table 5-8 Dlt_SendLogMessage 32
Table 5-9 Dlt_SetState 33
Table 5-10 Dlt_GetState 33
Table 5-11 Services used by the DLT 34
Table 6-1 Glossary 35
Table 6-2 Abbreviations 35

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.0	First release of the DLT module. This version implements only reporting of DET errors. Other functionalities as specified are not supported. Manual configuration.
1.1	Added reporting of DEM errors and logging of verbose and non-verbose messages in AUTOSAR4 use case.

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module DLT as specified in [1].

Supported AUTOSAR Release*:		
Supported Configuration Variants:	pre-compile	
Vendor ID:	DLT_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	DLT_MODULE_ID	55 decimal

* For the precise AUTOSAR Release please see the release specific documentation.

AUTOSAR 4.x introduced a new module called DLT which stands for Diagnostic Log and Trace. This module is used in order to trace messages from DET, DEM and SWCs. According to AUTOSAR the reported messages have to be displayed as text messages to the user. This is done by means of a PC tool – in case of Vector it is either CANoe or CANape.

While AUTOSAR defines a specific communication protocol to send trace messages from the ECU to the PC tool Vector's DLT implementation uses XCP.

2.1 Architecture Overview

The following figure shows where the DLT is located in the AUTOSAR architecture.

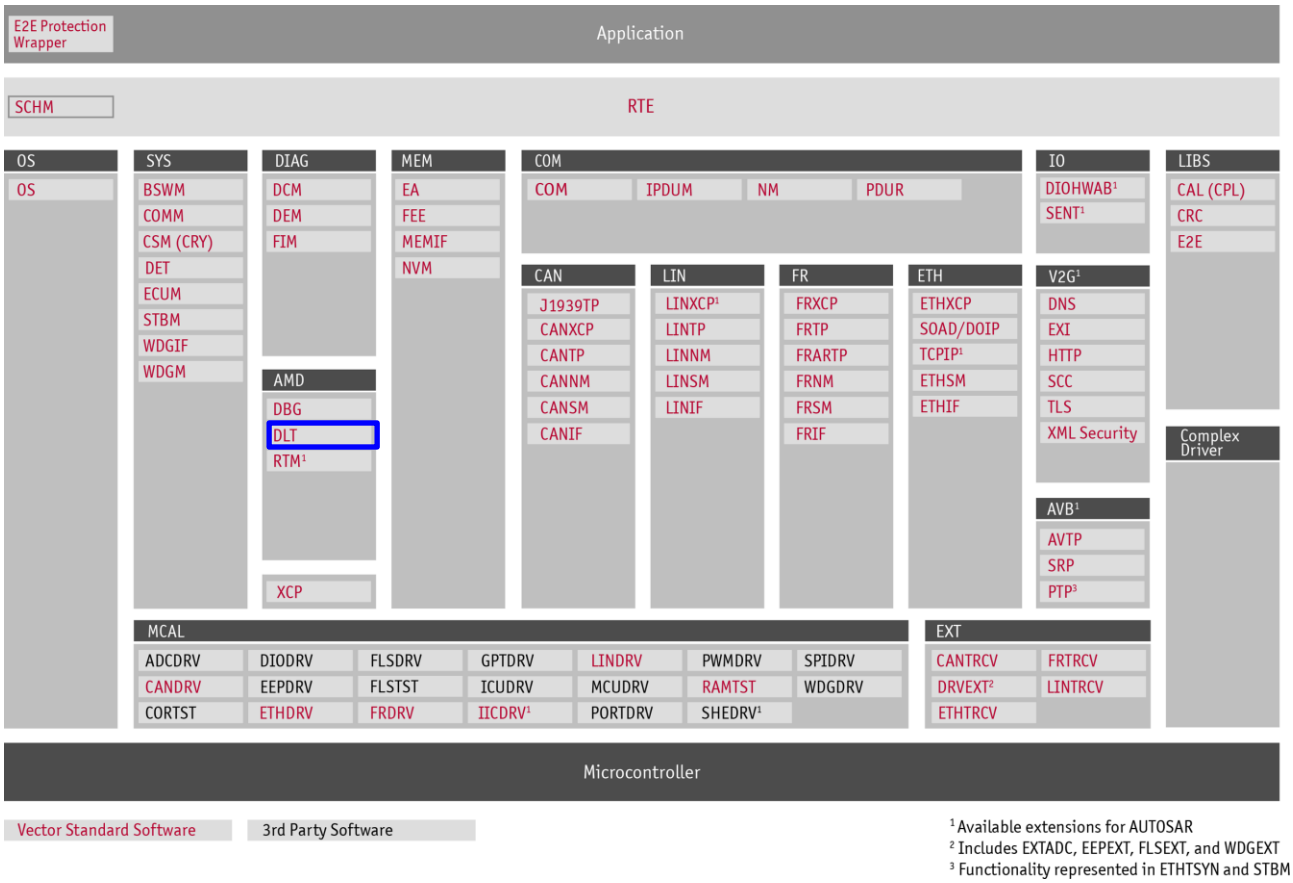


Figure 2-1 AUTOSAR 4.x Architecture Overview

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the DLT.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Deviations from AUTOSAR standard

3.1.1 Supported features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Security mechanism: explicit Dlt de-/activation.
Log messages from DET.
Log messages from DEM.
Verbose logging mode.
Non-verbose logging mode.
Logging of errors, warnings and info messages from AUTOSAR SWCs, providing a standardized AUTOSAR interface.
Gather all log and trace messages from all AUTOSAR SWCs in a centralized AUTOSAR service component (DLT) in BSW.

Table 3-1 Supported AUTOSAR standard conform features

3.1.2 Deviations

The following features specified in [1] are not supported:

Category	Description	Version
Functional	For DLT on XCP: Runtime configuration of DLT.	4.0.3
Functional	For DLT on XCP: Enable/disable individual log and trace messages.	4.0.3
Functional	For DLT on XCP: Messages for non-verbose logging are stored in an A2L fragment file instead of FIBEX.	4.0.3
Functional	For DLT on XCP: Communication via DltCom or DCM (instead XCP is used)	4.0.3
API	For DLT on XCP: The APIs Dlt_SendTraceMessage and Dlt_RegisterContext are not provided.	4.0.3
Functional	Communication over standard Dcm channel	4.0.3
Functional	RTE/VFB tracing (must be implemented manually and the hook functions are not generated)	4.0.3
Functional	Injection calls of SWCs.	4.0.3
Functional	Persistent storage of DLT configuration.	4.0.3

Category	Description	Version
Functional	Timing messages.	4.0.3
Functional	Bandwidth management.	4.0.3

Table 3-2 Deviations from AUTOSAR standard

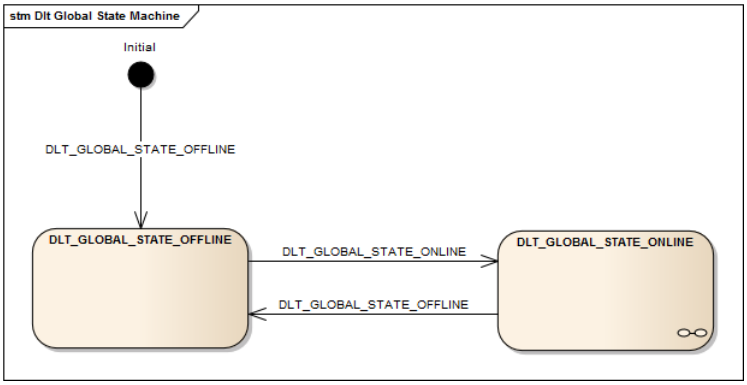
3.2 Initialization

The DLT module is pre-initialized by a call to function Dlt_InitMemory. The initialization includes all global data required to log data from ECU start on.

The DLT module is initialized by a call to the function Dlt_Init. Initialization of the DLT must not be done before the XCP driver is initialized and the corresponding bus interfaces and drivers. The function has a pointer parameter which is not used in the current version. Therefore the user should pass a NULL pointer.

3.3 States

There are two states of the global Dlt state machine. After Dlt_Init() the Dlt is in state DLT_GLOBAL_STATE_OFFLINE. Within this state only a subset of all Dlt services is available.



With the API Dlt_SetState() a state change can be requested. The next call to Dlt_MainFunction() changes the state as requested.

In state DLT_GLOBAL_STATE_ONLINE all Dlt services are available.

3.4 Main Functions

In case of XCP as protocol layer, the main function only handles the global state machine.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `DLT_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported DLT ID is 55.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x01	Dlt_Init
0x02	Dlt_GetVersionInfo
0x07	Dlt_DetForwardErrorTrace
0x15	Dlt_DemTriggerOnEventStatus
0x03	Dlt_SendLogMessage
0x50	Dlt_MainFunction
0x51	Dlt_SetState
0x52	Dlt_GetState

Table 3-3 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
0x01	DLT_E_WRONG_PARAMETERS	API service called with wrong parameter.
0x02	DLT_E_ERROR_IN_PROV_SERVICE	Provided API services of other modules returned with an error.
0x03	DLT_E_COM_FAILURE	The DLT communication module detects an error in communication with its external interfaces.
0x04	DLT_E_ERROR_TO_MANY_CONTEXT	Too many contexts are registered with the DLT module. (e.g. the configuration tables are full)
0x05	DLT_E_MSG_LOOSE	The internal message buffer is full and the oldest messages are overwritten.
0x06	DLT_E_PARAM_POINTER	API service called with a NULL pointer. In case of this error, the API service shall return immediately without any further action, besides reporting this development error.
0x07	DLT_E_INIT_FAILED	API was unable to initialize the service.
0x08	DLT_E_UNINITIALIZED	The DLT is not initialized, thus the requested Service is not available.
0x09	DLT_E_INVALID_STATE	The DLT is in an invalid global state.

Table 3-4 Errors reported to DET

3.5.1.1 Parameter Checking

The following table shows which parameter checks are performed on which services:

Service \ Check	DLT_E_WRONG_PARAMETERS	DLT_E_ERROR_IN_PROV_SERVICE	DLT_E_COM_FAILURE	DLT_E_ERROR_TO_MANY_CONTEXT	DLT_E_MSG_LOOSE	DLT_E_PARAM_POINTER	DLT_E_INIT_FAILED	DLT_E_UNINITIALIZED	DLT_E_INVALID_STATE
Dlt_Init									
Dlt_GetVersionInfo						■			
Dlt_DetForwardErrorTrace					■				
Dlt_DemTriggerOnEventStatus	■				■				
Dlt_SendLogMessage	■				■	■		■	
Dlt_MainFunction									
Dlt_SetState								■	
Dlt_GetState								■	

Table 3-5 Development Error Reporting: Assignment of checks to services

3.5.2 Production Code Error Reporting

The DLT module does not report Production Code Errors to the DEM.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR DLT into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the DLT contains the files which are described in the chapters 4.1.1 and 4.1.2. The files are located in the subfolder external\BSW\Dlt of your delivery.

4.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
Dlt.c	■	Delivered as library	This is the source file of the DLT module.
Dlt.h	■	■	This is the header file of the DLT module.
Dlt_Types.h	■	■	Defines required data types which are not defined by RTE.
_Dlt_Cfg.h	■	■	This file is a template. It contains the configuration of the DLT module.
_SchM_Dlt.h	■	■	This file is a template. It contains the SchM specific configuration for the DLT module.

Table 4-1 Static files

The files described in Table 4-1 must be included into your project. The source files must be included to your compiler and linker list, while the location of the header files must be added to the search path of the compiler.

Configuration of the DLT module is simply done by means of the template file _dlt_cfg.h. This file shall be copied and renamed into dlt_cfg.h. The file _SchM_Dlt.h shall also be renamed into SchM_Dlt.h.

Configuration Switch	Description
DLT_USE_COMLAYER_XCP	This define is set to STD_ON if XCP shall be used for communication of the DLT module with the XCP master tool.
DLT_USE_COMLAYER_ASR	This define is set to STD_ON if DltCom shall be used for communication of the DLT module with the Ethernet master tool.
DLT_DEV_ERROR_DETECT	If this define is set to STD_ON the DLT module checks development error sources. Since the DLT reports every DET error to the XCP master tool the actual reporting function Dlt_DetForwardErrorTrace does not report any DET errors in order to avoid recursion.
DLT_DEV_ERROR_REPORT	If this define is set to STD_ON the DLT module reports DET errors in case its checks fail. Since the DLT reports every DET error to the XCP master tool

Configuration Switch	Description
	the actual reporting function Dlt_DetForwardErrorTrace does not report any DET errors in order to avoid recursion.
DLT_VERSION_INFO_API	If this define is set to STD_ON the function Dlt_GetVersionInfo is available to query the version of the DLT module.
DLT_DET_EVENT	This define specifies the XCP event channel number which is used to report DLT messages to the XCP master tool. The default value is 60 (decimal).
DLT_DEM_EVENT_FILTERING_ENABLED	If this define is set to STD_ON, there have to be callback functions, implemented by the user. These callback functions are to filter DEM events (e.g. reject call to Dlt_DemTriggerOnEventStatus if previous status is equal to new status).
DLT_USE_VERBOSE_MODE	If this define is set to STD_ON, the timestamp is set in standard header of DLT messages. This define corresponds to field WTMS (With Timestamp).
DLT_MAX_MESSAGE_LENGTH	The maximum length of a DLT log or trace message (header + payload).

Table 4-2 Available configuration options of the DLT module

Table 4-2 describes the available configuration options which can be set in the file dlt_cfg.h. You need to adapt these options to your needs.

The second file which is delivered as a template is _SchM_Dlt.h. This file also needs to be copied and renamed into SchM_Dlt.h. You need to configure the following two defines to your needs and remove the #error directive from the file.

- SchM_Enter_Dlt
- SchM_Leave_Dlt

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File name	Description
Dlt_Cfg.h	Generated header file for pre-compile time configuration data.
Dlt_Cbk.c	Generated source file for SWC callback data.
Dlt_Cbk.h	Generated header file for SWC callback and pre-compile configuration data.
DltCom.c	Generated source file for SoAd callback data.
DltCom.h	Generated header file for SoAd callback data.

Table 4-3 Generated files

4.2 Include Structure

Figure 4-1 shows the include structure of the DLT module. In order to access DLT API functions your C files need to include the file Dlt.h.

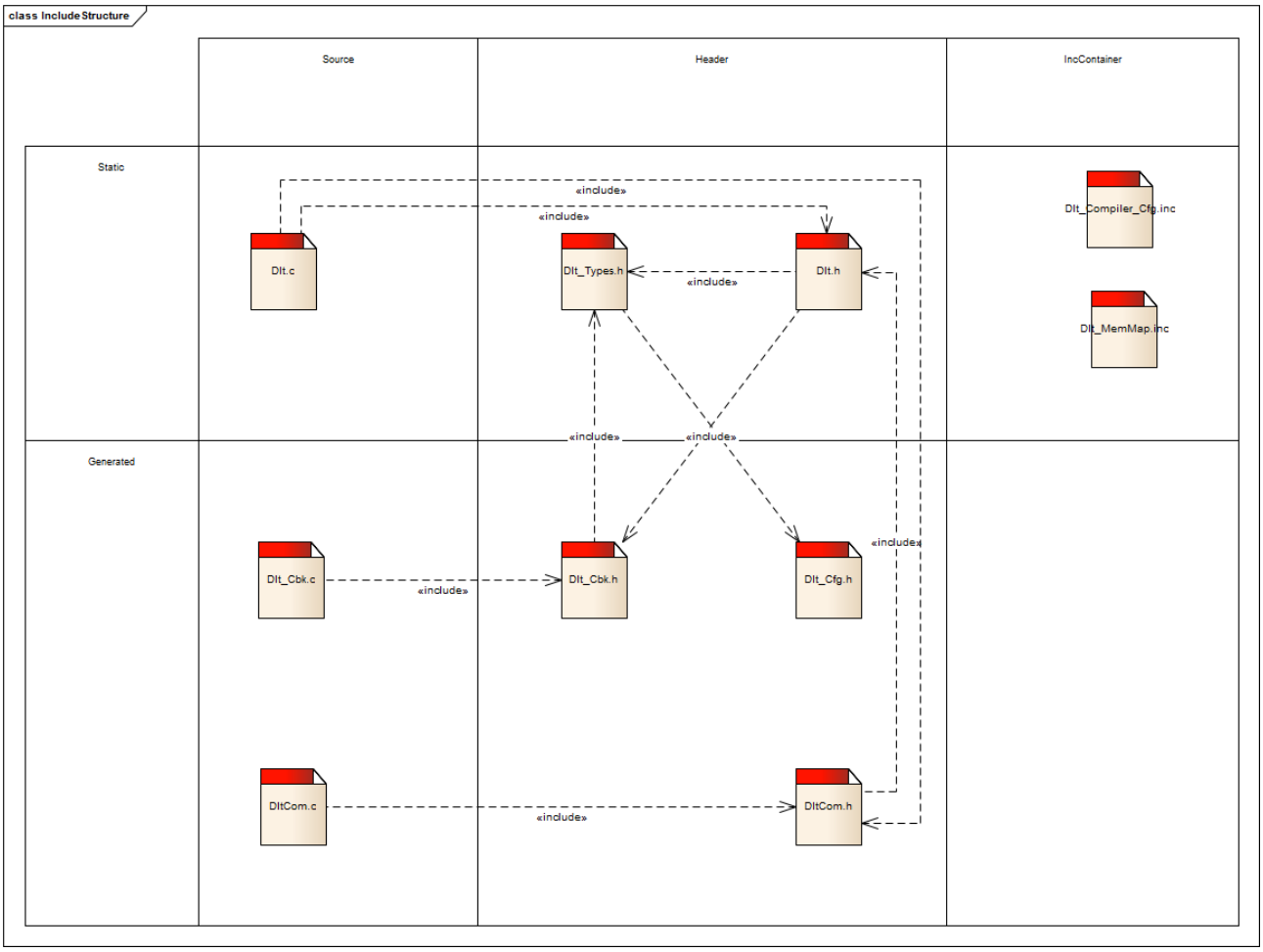


Figure 4-1 Include structure DLT

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions of the DLT and illustrates their assignment among each other.

Memory Mapping Sections / Compiler Abstraction Definitions	DLT_CODE	DLT_CONST	DLT_VAR_INIT	DLT_VAR_NOINIT	DLT_VAR_ZERO_INIT
DLT_START_SEC_CONST_8BIT, DLT_STOP_SEC_CONST_8BIT		■			
DLT_START_SEC_CONST_UNSPECIFIED, DLT_STOP_SEC_CONST_UNSPECIFIED		■			
DLT_START_SEC_VAR_NOINIT_32BIT DLT_STOP_SEC_VAR_NOINIT_32BIT				■	
DLT_START_SEC_VAR_NOINIT_UNSPECIFIED DLT_STOP_SEC_VAR_NOINIT_UNSPECIFIED				■	
DLT_START_SEC_VAR_ZERO_INIT_UNSPECIFIED DLT_STOP_SEC_VAR_ZERO_INIT_UNSPECIFIED			■		
DLT_START_SEC_VAR_INIT_32BIT DLT_STOP_SEC_VAR_INIT_32BIT			■		
DLT_START_SEC_VAR_INIT_UNSPECIFIED DLT_STOP_SEC_VAR_INIT_UNSPECIFIED			■		
DLT_START_SEC_CODE DLT_STOP_SEC_CODE	■				

Table 4-4 Compiler abstraction and memory mapping

4.4 Critical Sections

The DLT module uses just one single exclusive area which is entered in case the DLT computes the message code to be transmitted to the XCP master tool.

Exclusive Area	Description
DLT_EXCLUSIVE_AREA_0	Protects the computation of the reported message code. This exclusive area shall protect the function Dlt_DetForwardErrorTrace by interruption by itself. Since it is called from all possible contexts in the BSW a global lock is the safest way of configuration. The runtime of this exclusive area is very short. It includes less than 10 code lines without any function calls.

Table 4-5 Exclusive Areas of the DLT module

4.5 Common Configuration Steps

The DLT provides services to change and request ECU internal data. To avoid misuse of these services, the DLT provides a security mechanism. After initialization the DLT is in

state DLT_GLOBAL_STATE_OFFLINE, where most services, like communication to external client, are not available.

The activation of DLT has to be requested explicitly via a call of Dlt_SetState(DLT_GLOBAL_STATE_ONLINE). With next call of Dlt_MainFunction all services of DLT are available. To grant no misuse, the call of Dlt_SetState should be in context of a diagnostic session.

With a call of Dlt_SetState(DLT_GLOBAL_STATE_OFFLINE) the DLT is deactivated again. All stored messages are removed from DLT buffers.

These APIs have to be called from the application. Therefore the RTE provides interfaces for the application to access these APIs indirectly.

Caution
It is highly recommended to call the APIs Dlt_SetState and Dlt_GetState in context of a diagnostic session.

4.6 DLT on XCP

There are two versions of DLT for AUTOSAR 4; DLT on XCP and AUTOSAR DLT. The difference is the communication layer used and the feature coverage. DLT on XCP uses XCP as communication protocol and supports a subset of features.

4.6.1 XCP Event

The DLT module uses XCP for data transmission to the PC tool which provides a GUI to display the reported messages.

There are four XCP events that can be triggered by DLT. Table 4-6 shows these events and the functions where they are triggered.

Function	XCP event
Dlt_SendLogMessage	DLT_NON_VERBOSE_MSG_EVENT DLT_VERBOSE_MSG_EVENT
Dlt_DemTriggerOnEventStatus	DLT_DEM_EVENT
Dlt_DetForwardErrorTrace	DLT_DET_EVENT

Table 4-6 DLT functions and their corresponding XCP events

Each time one of the functions in Table 4-6 is called the DLT module calls the corresponding XCP event to trigger transmission of the message.

The default value is set to 60 (decimal). In case you wish to change this value the change needs to be done also in the file DltEvents.a2l.

Note
Your XCP driver needs to be configured in a way that XCP events are used.

4.6.2 XCP: Logging of verbose and non-verbose DLT messages

The DLT supports the verbose and non-verbose transmission mode for log messages.

> Non-Verbose Mode:

In the Non-Verbose Mode the XCP master measures only the Message ID. This Message ID is unique for a specific DLT message and the static message text is associated to this ID. The mapping between Message ID and static text is provided by the generated A2L file.

Example
Non-verbose DLT messages can be transmitted as following:

```
Dlt_NonVerboseMsgType nonVerboseMsg = { {0, 0, 0 }, DltConf_DltNonVerboseMessage_Msg1};
Dlt_MessageLogInfoType msgLogInfoType = { DLT_LOG_ERROR, DLT_NON_VERBOSE_MSG, 0, 0};

retVal = Dlt SendLogMessage( 0, &msgLogInfoType, (P2CONST(uint8, AUTOMATIC,
DLT_APPL_VAR)) &nonVerboseMsg, 0 );
```

Expert Knowledge
Instead of using an integer value as Message ID when calling Dlt_SendLogMessage() you can use the generated Symbolic Name Value define like in the example above. The generated define depends on the short name of the corresponding DltNonVerboseMessage container (marked red in the example).

This way the identification of static messages is easier.

The corresponding static message can be configured as depicted below:

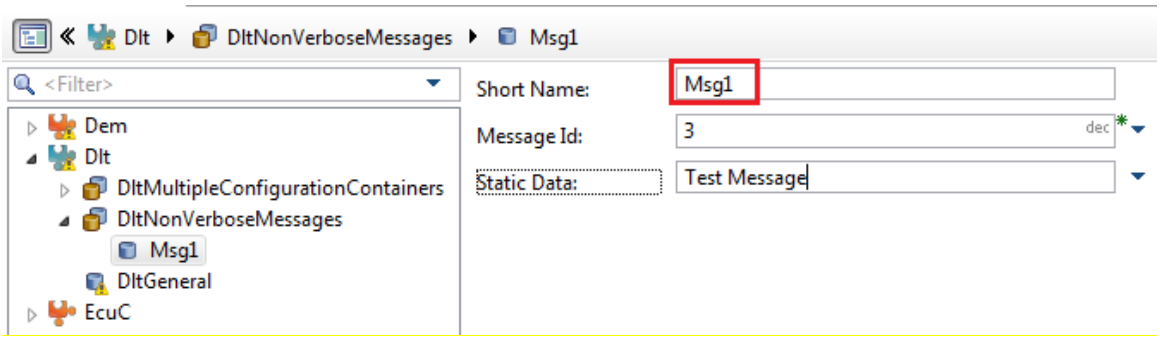


Figure 4-2 Configuration of non-verbose messages

> **Verbose Mode:**

The Verbose Mode is supported if DLT_USE_VERBOSE_MODE is STD_ON. In contrary to the Non-Verbose Mode, XCP master measures and displays the complete non-static text in the Verbose Mode. The maximum allowed message length is defined by the preprocessor switch DLT_MAX_MESSAGE_LENGTH.

Example
Verbose DLT messages can be transmitted as following:

```
Dlt VerboseMsgType verboseMsg = { {0, 0, 0 }, 3 /* MessageId */, "Test Message"};
Dlt_MessageLogInfoType msgLogInfoType = { DLT_LOG_INFO, DLT_VERBOSE_MSG, 0, 0};


retVal = Dlt SendLogMessage( 0, &msgLogInfoType, (P2CONST(uint8, AUTOMATIC,
DLT APPL VAR)) &verboseMsg, sizeof("Test Message\n"));
```

or using sprintf():

```
Dlt VerboseMsgType verboseMsg = { {0, 0, 0 }, 3 /* MessageId */};
Dlt MessageLogInfoType msgLogInfoType = { DLT LOG INFO, DLT VERBOSE MSG, 0, 0};
uint16 len = sprintf ( verboseMsg.Payload, "The half of %d is %d", 60, 60/2 );
retVal = Dlt_SendLogMessage( 0, &msgLogInfoType, (P2CONST(uint8, AUTOMATIC,
DLT APPL VAR)) &verboseMsg, len);
```

Caution
The DLT_MAX_MESSAGE_LENGTH parameter must be configured with caution because this parameter directly affects the runtime of the Dlt_SendLogMessage() function when logging verbose messages.

4.6.3 Workflow McData

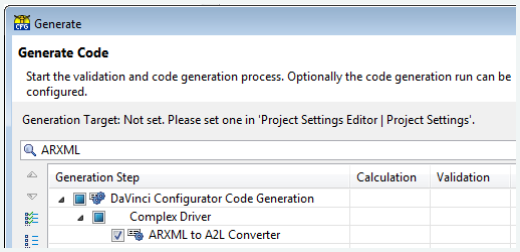
To use any AMD module (DBG, DLT or RTM) with XCP as protocol layer, XCP requires the mapping between symbols and memory addresses. For this purpose there is a master a2l file. This a2l file can then be used by CANoe or CANape to stimulate and/or acquire data from ECU.

Practical Procedure

1. Copy .\external\Misc\McData_master.a2l to .\Config\McData\ a rename it to master.a2l.

2. Open it with an editor and edit all lines with “TODO” as required for your purpose.

3. Open your config with DaVinci Configurator. There is a generator called “ARXML to A2L Converter”. This generator has no configurable content, it just has to be generated.



This converter creates the McData.a2l and McDataExport.arxml. These files contain all data required for the symbols which will later be available in CANoe.

4. Switch to .\CANoe and create the init file Updater.ini if it does not exist. Open it in an editor and add following lines:

```
[ELF]
ELF_ARRAY_INDEX_FORM=1
MAP_MAX_ARRAY=100

[UBROF]
UBROF_ARRAY_INDEX_FORM=1
MAP_MAX_ARRAY=70

[OMF]
OMF_ARRAY_INDEX_FORM=1
MAP_MAX_ARRAY=70

[PDB]
PDB_ARRAY_INDEX_FORM=1
MAP_MAX_ARRAY=70

[OPTIONS]
FILTER_MODE=1
MAP_FORMAT=31  ;Refer to ASAP Updater User Manual for description of
MAP file format
```

The bold values should be individually adapted. The MAP_MAX_ARRAY defines how big the maximum array can be that is read/written by XCP.

The MAP_FORMAT has to be set to according to the used map file. For example set it to 31 if you use an elf file (32bit CPU). Set it to 54 if you are using CANoe.Emu. For other values please refer to [5].

If your project is compiled the a2l updater has to be executed. If you have no updater yet, create a batch file called update_a2l.bat in the directory where your map file is. Open it with an editor. The content should include the absolute path to the ASAP2Updater.exe, and the relative paths to:

- the input file (Master.a2l)
- the output file (AmdResult.a2l)

> the map file (<ProjectName>.<[elf|pdb|map|…]>)

> the Update.ini file

> and optionally an log file (AsapUpdater.log)

For example this could look like:

"C:\Program Files (x86)\Vector CANwin 8.5\ASAP2Updater\Exec\ASAP2Updater.exe" -I ..\Config\McData_Master.a2l -A .\Tsp_BAC4_REF_VC121.elf -O ..\Canoe\AmdResult.a2l -T ..\Canoe\Updater.ini -L ..\Log\AsapUpdater.log

The updater uses the Master a2l, containing the symbols, and the map file to map all symbols to memory addresses.

XCP can now use the result a2l.

4.6.4 DLT Reporting with CANoe

Note
You need a license of CANoe's XCP option in order to use it as a XCP master

CANoe provides no map updater. Therefore it is necessary to run an address update of the A2L files using external updater tools (like Vector's ASAP Toolset). Afterwards you can load the resulting A2L file into CANoe by selecting XCP/CCP Configuration in the CANoe's Configuration menu.

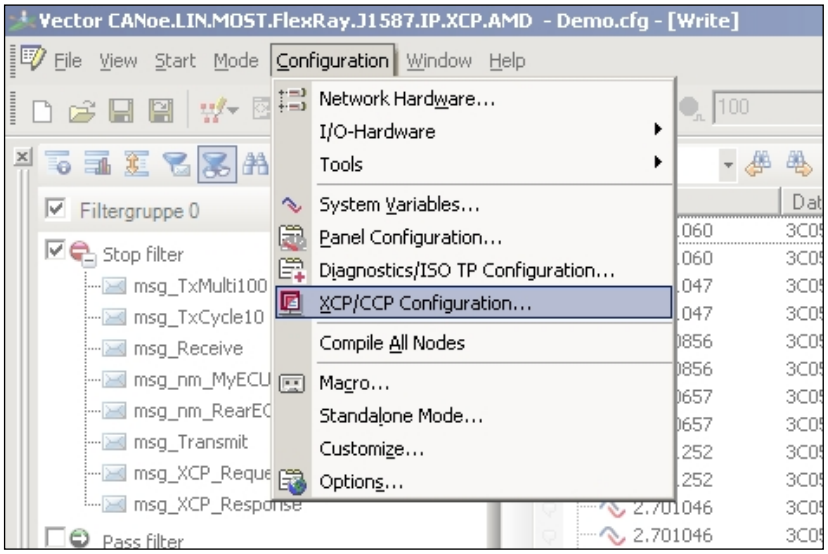


Figure 4-3 Open XCP/CCP option

> Add the master A2L which includes the DLT A2L fragments via the [Add…] button to the XCP configuration.

- DLT Reporting of DET Errors: Select the “Signal Configuration” tab and add the signal Dlt_DetErrorCode to your measurement list. Select DLT_DET as XCP event channel for measurement as shown in Figure 4-4.
- DLT Reporting of DEM Event Status Changes: Select the “Signal Configuration” tab and add the signal Dlt_DemEventStatus to your measurement list. Select DLT_DEM as XCP event channel for measurement.
- DLT Logging of Non-Verbose Messages: Select the “Signal Configuration” tab and add the signal Dlt_NonVerboseMessageId to your measurement list. Select DLT_MSG as XCP event channel for measurement.
- DLT Logging of Verbose Messages: Select the “Signal Configuration” tab and add the signal Dlt_VerboseMessageData to your measurement list. Select DLT_VMSG as XCP event channel for measurement.

Note
DLT Logging of Verbose Messages requires at least CANoe 8.1 SP3 otherwise the measured data will be displayed as raw value instead of string.

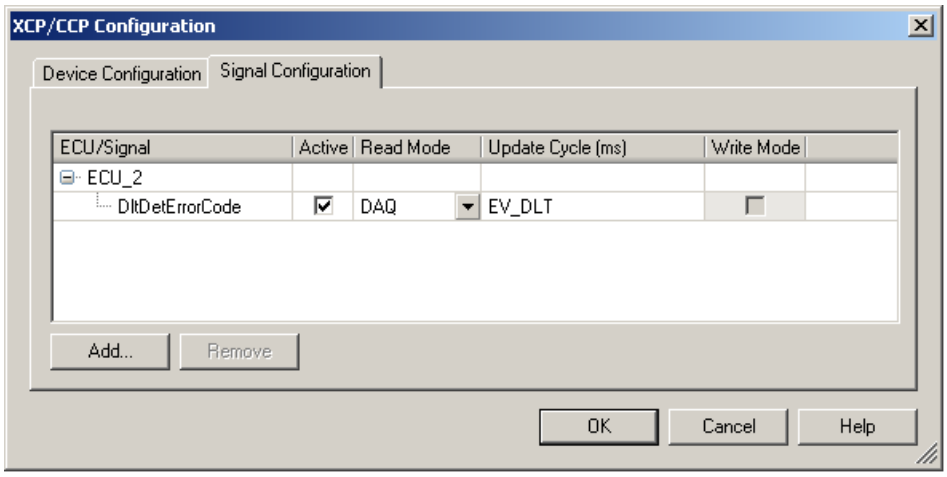


Figure 4-4 Selection of the XCP measurement object for DLT logging of DET reports

In order to display the reported DLT messages you can add a new trace window to your CANoe configuration and filter out all messages and events except for example Dlt_DetErrorCode. It is important to select the [sym] button in the toolbar in order to enable the output of textual DLT messages into CANoe’s trace window.

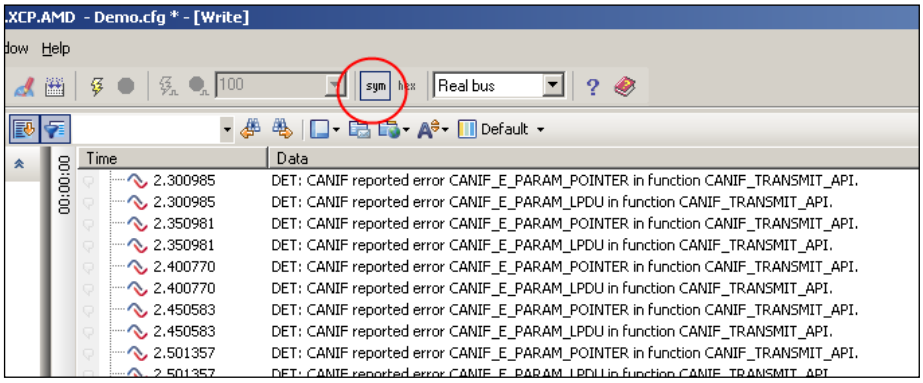


Figure 4-5 Enable the symbolic option by clicking the button [sym] in CANoe's toolbar.

Note

The raw values (32bit in hex) of reported DET errors can be interpreted as following:

0x II : Instance ID
MM : Module ID
SS : Service ID
EE : Error ID

E.g: 0x00380202 /* Instance 0x00,
Module ID = 0x38 -> "SoAd",
Service ID = 0x02 -> "SoAd_GetVersionInfo",
Error ID = 0x02 -> "SOAD_E_PARAM_POINTER" */

4.6.5 DLT Reporting with CANape

In order to log DLT messages with CANape the file Dlt.A2L has to be included into a master A2L file. The addresses in the A2L file can be updated using CANape's map updater. Once the A2L file is incorporated into the CANape configuration one can select the DLT measurement object using CANape's symbol explorer. The DLT variable for measurement can be found in the DLT folder in the A2L file as shown in Figure 4-6.

- DLT Reporting of DET Errors: Select the variable Dlt_DetErrorCode and add it to a text window as shown in Figure 4-7.
- DLT Reporting of DEM Event Status Changes: Select the variable Dlt_DemEventStatus and add it to a text window.
- DLT Logging of Non-Verbose Messages: Select the variable Dlt_NonVerboseMessageId and add it to a text window.
- DLT Logging of Verbose Messages: Select the variable Dlt_VerboseMessageData and add it to a text window.

DLT messages can be displayed using CANape's text window (Figure 4-8).

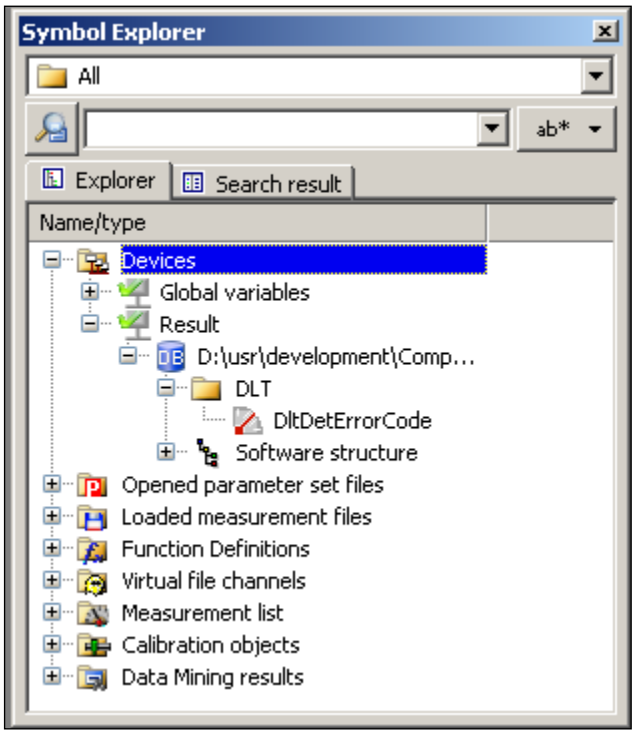


Figure 4-6 CANape's symbol explorer for selection of the DLT variable

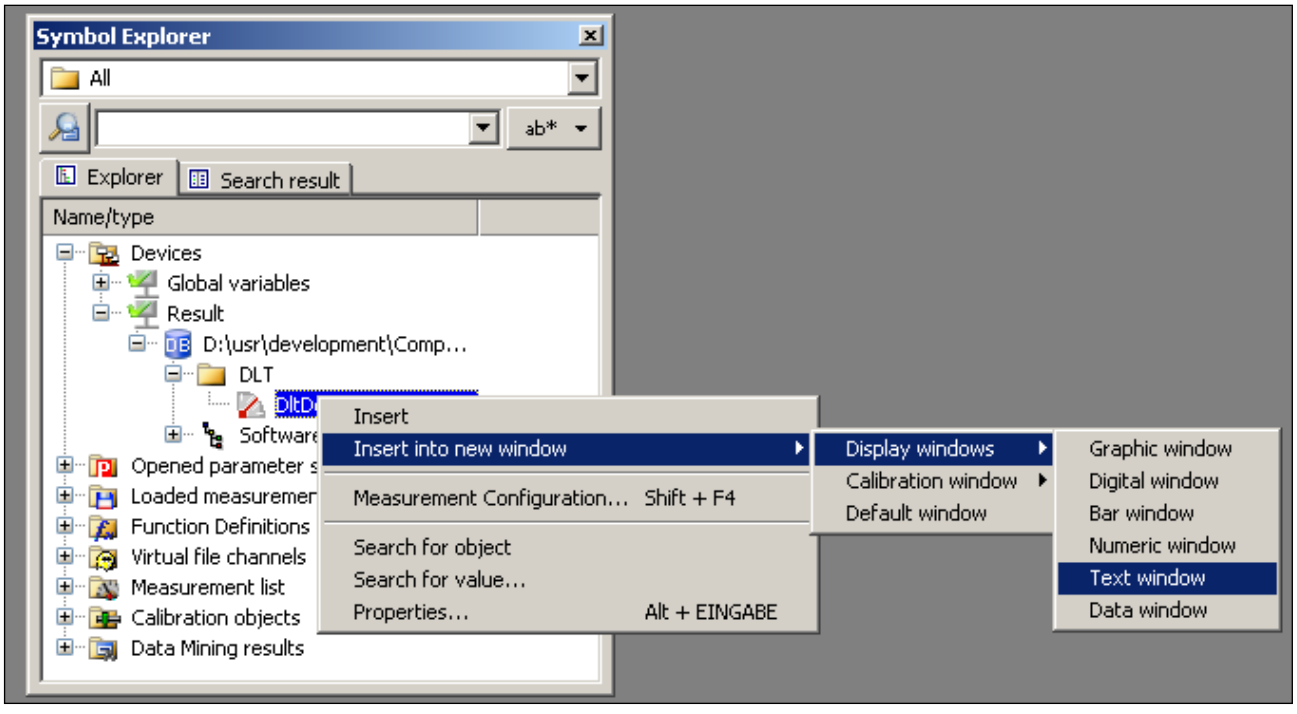


Figure 4-7 Select the DLT variable for measurement in a text window

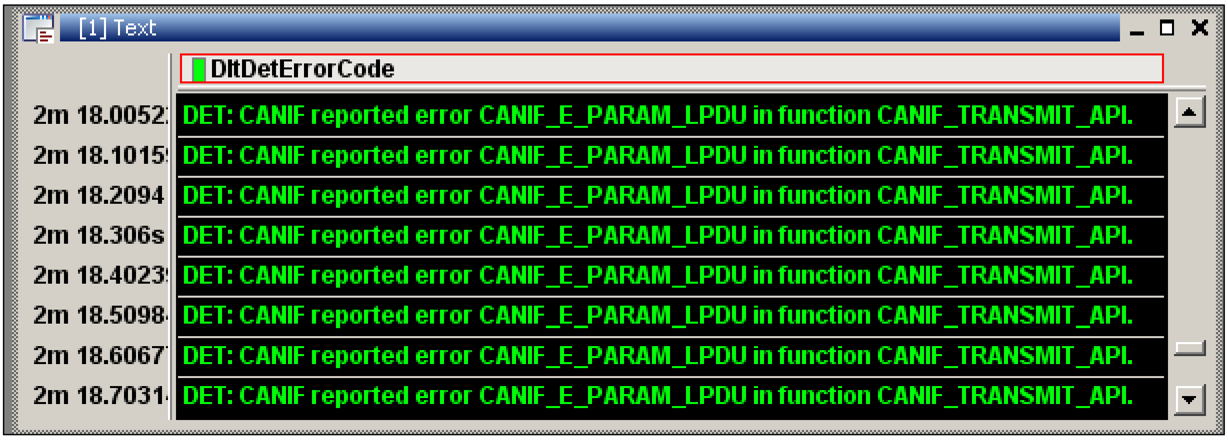


Figure 4-8 Logging of DLT messages using CANape's text window.

5 API Description

5.1 Type Definitions

The types defined by the DLT are described in this chapter.

Type Name	C-Type	Description	Value Range
Dlt_ConfigType	uint8	This type is currently not used by the DLT module. It exists to satisfy the needs of the Dlt_Init function.	n.a
Dlt_SessionIDType	uint32	This type describes the Session ID.	n.a.
Dlt_ApplicationIDType	uint32	This type describes the Application ID.	n.a.
Dlt_MessageIDType	uint32	This type describes the unique Message ID for a message.	n.a.
Dlt_MessageOptionsType	uint8	This type determines the message type (log or trace) and whether verbose mode is used or not.	n.a.
Dlt_MessageLogLevelType	Enum	This type describes the log level for each log message.	0..6
Dlt_MessageLogInfoType	Struct	This type describes the message log level, message options, context ID and application ID of a verbose or non-verbose message.	n.a.
Dlt_MessageTraceInfoType	Struct	This type describes the message trace status, message options, context ID and application ID of a verbose or non-verbose message	n.a

Type Name	C-Type	Description	Value Range
Dlt_MessageTraceType	Enum	This type describes the trace status for each trace message.	1..5
Dlt_ReturnType	Enum	This type describes the return values of the DLT services.	0..7

Table 5-1 Type definitions

5.2 Services provided by DLT

5.2.1 Dlt_InitMemory

Prototype	
`void` **`Dlt_InitMemory`** `(void)`	
Parameter	
`void`	-
Return code	
`void`	-
Functional Description	
The global data of DLT module is initialized by calling this function. This function must be called before Dlt_Init.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This function shall be called on task level.	

Table 5-2 Dlt_InitMemory

5.2.2 Dlt_Init

Prototype	
`void` **`Dlt_Init`** `(const Dlt_ConfigType* ConfigPtr)`	
Parameter	
`ConfigPtr`	Pointer to a configuration structure for DLT module initialization. Currently this parameter is unused and a NULL pointer can be passed.
Return code	
`void`	-

Functional Description
The DLT module is initialized by calling this function. This function must be called before any other function of the DLT is called.
Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > This function shall be called after initialization of the communication drivers, interfaces and the XCP module.
Expected Caller Context
> This function shall be called on task level.

Table 5-3 Dlt_Init

5.2.3 Dlt_MainFunction

Prototype	
`void` **`Dlt_MainFunction`** `(void)`	
Parameter	
`void`	-
Return code	
`void`	-
Functional Description	
The main function is empty.	
Particularities and Limitations	
> The main function exists despite the requirement [Dlt468]. > This function is synchronous. > This function is non-reentrant. > This function shall be called cyclically by RTE.	
Expected Caller Context	
> This function shall be called on task level.	

Table 5-4 Dlt_MainFunction

5.2.4 Dlt_GetVersionInfo

Prototype	
`void` **`Dlt_GetVersionInfo`** `(Std_VersionInfoType* VersionInfo)`	
Parameter	
`VersionInfo`	Pointer to a RAM structure which is initialized with the version number of the DLT module.
Return code	
`void`	-

Functional Description
This function writes the DLT module version into the structure referenced by the given pointer parameter.
Particularities and Limitations
> This function is only available if the preprocessor switch DLT_VERSION_INFO_API is set to STD_ON. > This function is non-reentrant. > This function is synchronous.
Call context
> No limitations

Table 5-5 Dlt_GetVersionInfo

5.2.5 Dlt_DetForwardErrorTrace

Prototype	
`void` **`Dlt_DetForwardErrorTrace`** `(uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)`	
Parameter	
`ModuleId`	The module which called the function Det_ReportError
`InstanceId`	The instance of the calling module
`ApiId`	The identifier of the API where the DET error occurred
`ErrorId`	The number of the reported error
Return code	
`void`	-
Functional Description	
This function is called from within the context of the DET module function Det_ReportError.	
Particularities and Limitations	
> This function is non-reentrant. > This function is synchronous.	
Call context	
> The call context of the DET call. Every context in which BSW modules are called is possible.	

Table 5-6 Dlt_DetForwardErrorTrace

5.2.6 Dlt_DemTriggerOnEventStatus

Prototype
`void` **`Dlt_DemTriggerOnEventStatus`** `(Dem_EventIdType EventId, Dem_EventStatusExtendedType EventStatusOld, Dem_EventStatusExtendedType EventStatusNew)`

Parameter	
EventId	Identification of an Event by assigned event number. The Event Number is configured in the Dem. > Min.: 1 (0: Indication of no Event or Failure) > Max.: Result of configuration of Event Numbers in Dem (Max is either 255 or 65535)
EventStatusOld	Extended event status before change
EventStatusNew	Detected / reported of event status
Return code	
void	-
Functional Description	
This service is provided by the Dem in order to call DLT upon status changes. It is possible to filter DemEvent status changes by setting DLT_DEM_EVENT_FILTERING_ENABLED to STD_ON. In this case the DLT module calls the Appl_DltDemEventFilterCbk() function. Hence the application has the chance to filter specific DEM Events IDs or DEM Event Status Bits. If the preprocessor switch DLT_DEM_EVENT_FILTERING_ENABLED is STD_OFF all changes of all DEM Events are reported by the DLT.	
Particularities and Limitations	
> The call to the function Dlt_DemTriggerOnEventStatus must be configured in the DEM module's configuration. > This function is non-reentrant. > This function is synchronous.	
Call context	
> The call context of the DEM Event status change.	

Table 5-7 Dlt_DemTriggerOnEventStatus

5.2.7 Dlt_SendLogMessage

Prototype
`Dlt_ReturnType Dlt_SendLogMessage ( Dlt_SessionIDType SessionId, P2CONST(Dlt_MessageLogInfoType, AUTOMATIC, DLT_APPL_VAR) LogInfo, P2CONST(uint8, AUTOMATIC, DLT_APPL_VAR) LogData, uint16 LogDataLength)`

Parameter	
SessionId	For SW-C this is not visible (Port defined argument value), for BSW-modules it is the module number.
LogInfo	Structure containing information whether the message shall be transmitted in verbose or non-verbose mode and filtering information.
LogData	Buffer containing the parameters to be logged. The content of this pointer represents the payload of the send log message.

LogDataLength	Length of the data buffer LogData.
Return code	
Dlt_ReturnType	DLT_E_MSG_TOO_LARGE - The message is too large for sending over the network DLT_E_IF_NOT_AVAILABLE - The interface is not available. DLT_E_UNKNOWN_SESSION_ID - The provided session id is unknown. DLT_E_NOT_IN_VERBOSE_MODE - Unable to send the message in verbose mode. DLT_E_OK - The required operation succeeded.
Functional Description	
The service represents the interface to be used by basic software modules or by software component to send verbose and non-verbose log messages. For details how verbose or non-verbose DLT messages are transmitted refer to section 4.6.2.	
Particularities and Limitations	
> It is expected that the value LogInfo->log_info is in valid range (DLT_LOG_FATAL – DLT_LOG_VERBOSE). There is no check, thus the value is passed to external tool. > The Message IDs used for DEM (0x00000001) and DET (0x00000002) are reserved and not usable for non-verbose log messages. > Logging with verbose messages is only possible if DLT_USE_VERBOSE_MODE is STD_ON. > Transmitting a mixture of static and non-static data is not supported for non-verbose messages. > This function is non-reentrant. > This function is synchronous.	
Call context	
> This function can be called in any context.	

Table 5-8 Dlt_SendLogMessage

5.2.8 Dlt_SetState

Prototype	
`Std_Returntype Dlt_SetState (Dlt_GlobalStateType NewState)`	
Parameter	
NewState	The new global DLT state.
Return code	
Std_Returntype	E_OK: The global DLT state change was successfully requested. E_NOT_OK: The change request was rejected.

Functional Description
The service dis-/enables the DLT. In global DLT state DLT_GLOBAL_STATE_OFFLINE, most DLT services are not available. Whereas in global DLT state DLT_GLOBAL_STATE_ONLINE all DLT services are available.
Particularities and Limitations
> This function is non-reentrant. > This function is synchronous.
Call context
> This service should be called in context of a diagnostic session.

Table 5-9 Dlt_SetState

5.2.9 Dlt_GetState

Prototype	
`Std_Returntype Dlt_GetState (Dlt_GlobalStateType* CurrentStatePtr)`	
Parameter	
`CurrentStatePtr`	The requested global DLT state.
Return code	
`Std_Returntype`	E_OK: The return of current global DLT state was successfully. E_NOT_OK: The current global DLT state could not be returned.
Functional Description	
The service returns the current global DLT state.	
Particularities and Limitations	
> This function is non-reentrant. > This function is synchronous.	
Call context	
> This service should be called in context of a diagnostic session.	

Table 5-10 Dlt_GetState

5.3 Services used by DLT

In the following table services provided by other components are listed, which are used by the DLT. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
XCP	Xcp_Event
SCHM	SchM_Enter_Dlt

Component	API
	SchM_Exit_Dlt
APPL	Appl_DltDemEventFilterCbk

Table 5-11 Services used by the DLT

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
Cfg5	Generation tool for MICROSAR components
DltViewer	DLT master tool of GENIVI.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CD	Complex Driver
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DLT	Diagnostic Log and Trace
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SWC	Software Component
SWS	Software Specification
VFB	Virtual Function Bus

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com