VECTOR

MICROSAR Diagnostic Log and Trace Autosar

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

Illustrations

Tables

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.

Vector supports the communication protocol specified by AUTOSAR as well as XCP as communication protocol.

2.1 Architecture Overview

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

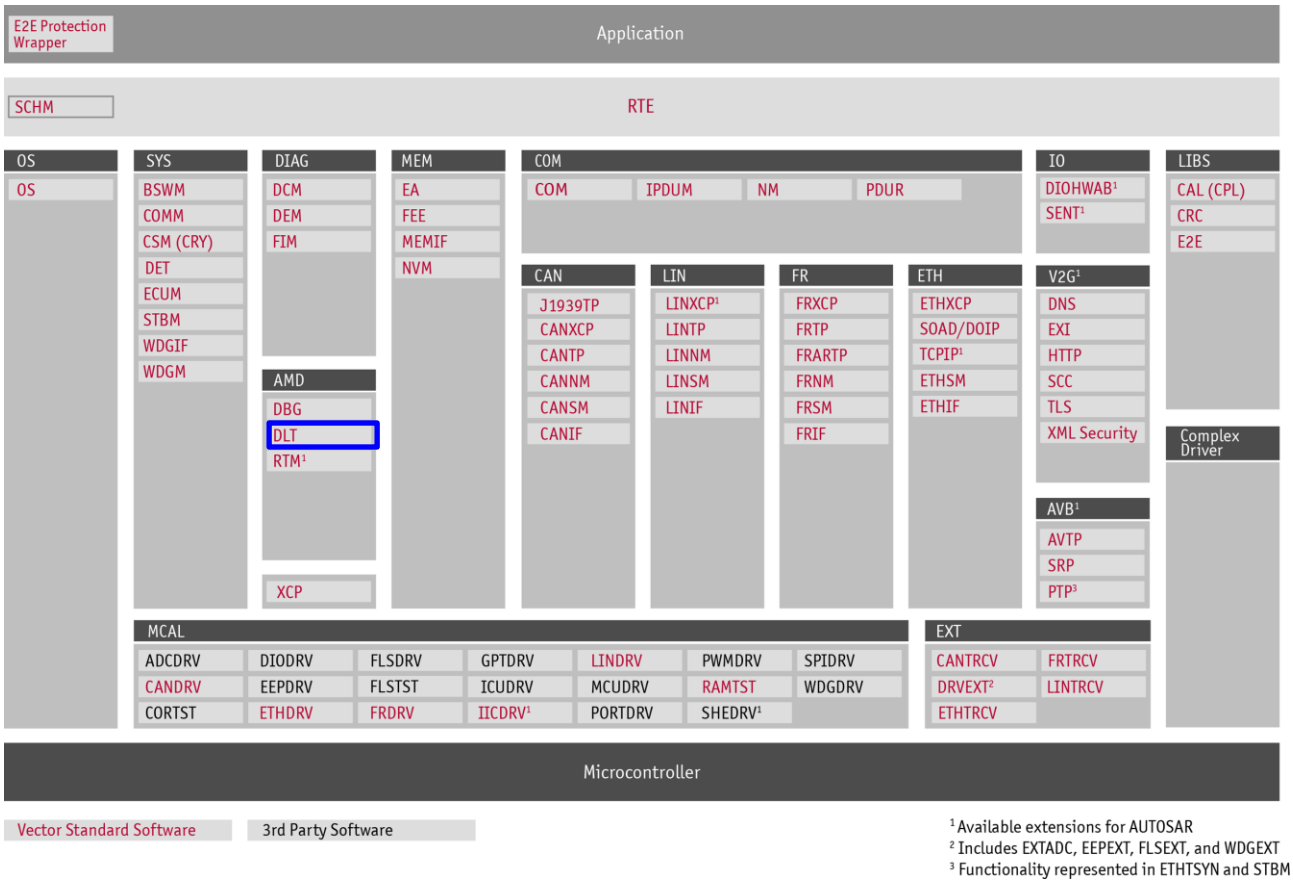


Figure 2-1 AUTOSAR 4.x Architecture Overview

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.
For DLT with DltCom: Runtime configuration of DLT module.
For DLT with DltCom: Enable/disable individual log and trace messages.

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

Category	Description	Version
Functional	Injection calls of SWCs.	4.0.3
Functional	Persistent storage of DLT configuration.	4.0.3
Functional	Timing messages.	4.0.3
Functional	Bandwidth management.	4.0.3
Functional	DLT completely event triggered (This DLT implementation uses a main function, therefore it has a receive buffer).	4.0.3
API	For DLT with DltCom: The following control services return `DLT_NOT_SUPPORTED` if requested: - Get_LogInfo - Store_Config - SetComInterfaceStatus - GetComInterfaceStatus - GetComInterfaceNames - SetComInterfaceMaxBandwidth - GetComInterfaceMaxBandwidth - SetTimingPackets - CallSW-Cinjection	4.0.3
API	For DLT with DltCom: The following APIs are not implemented: - DltCom_CancelTransmitRequest - DltCom_SetInterfaceStatus - Dlt_ConditionCheckRead - Dlt_WriteData - Dlt_ReadDataLength - Dlt_ReadData - Dlt_ActivateEvent	4.0.3
API	For DLT with DltCom: The APIs Dlt_SendLogMessage, Dlt_SendTraceMessage and Dlt_RegisterContext have return type Std_ReturnType instead of Dlt_ReturnType. Nevertheless, the values of Dlt_ReturnType are returned.	4.0.3 < 4.2.2
API	For DLT with DltCom: The following functions differ in their name: - DltCom_CopyRxData -> DltCom_ SoAdTpCopyRxData - DltCom_CopyTxData -> DltCom_SoAdTpCopyTxData - DltCom_StartOfReception -> DltCom_SoAdTpStartOfReception	4.1.2
API	For DLT with DltCom: In function DltCom_SoAdTpStartOfReception the type of parameter “info” differs to specification: - const PduInfoType * info -> PduInfoType * info	4.1.2

Table 3-2 Deviations from AUTOSAR standard

Note
The FIBEX file required for communication via DltCom is not generated, thus it has to be created manually.
How to create this file is described in chapter 4.7.8.

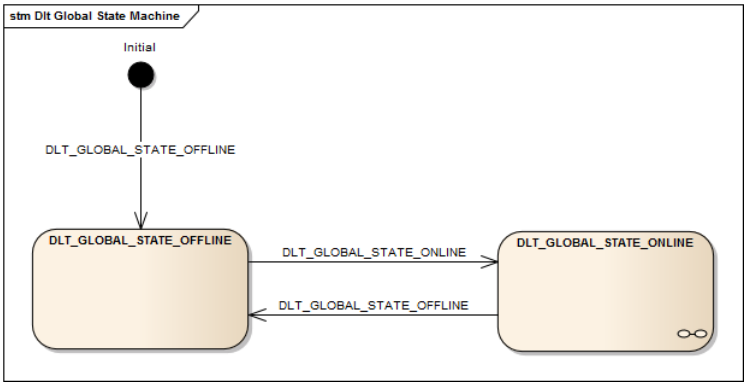
3.2 Initialization

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

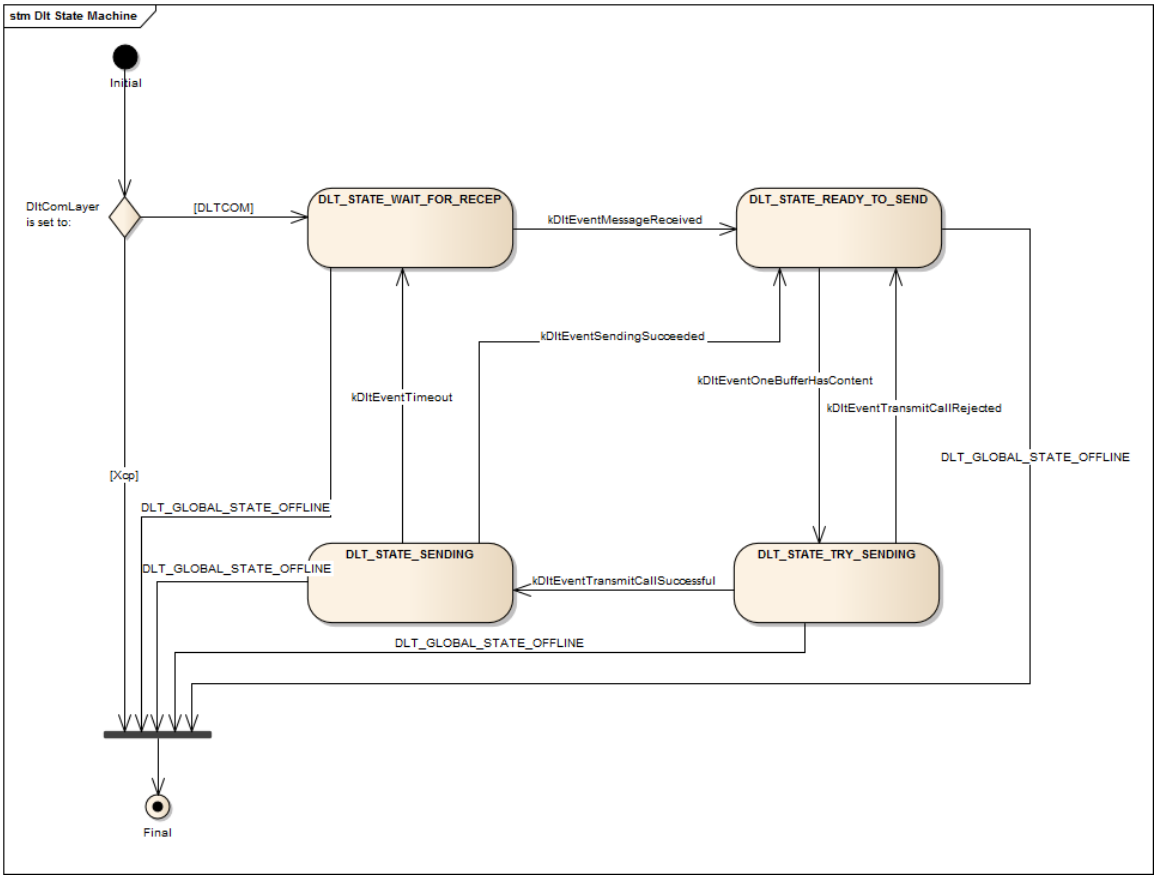
There are two states of the global Dlt state machine. After Dlt_Init() the Dlt is in state DLT_GLOBAL_STATE_OFFLINE. Within this state only a subset of all Dlt services is available.



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.

The DLT with DltCom supports a sub state machine within the global state DLT_GLOBAL_STATE_ONLINE.



The sub state machine of DLT with DltCom is described in the following table.

State	Next state	Transition (Event)
DLT_STATE_WAIT_FOR_RECEP	DLT_STATE_READY_TO_SEND	Message received (kDltEventMessageReceived)
DLT_STATE_READY_TO_SEND	DLT_STATE_TRY_SENDING	at least one buffer has content to send (kDltEventOneBufferHasContent)
DLT_STATE_TRY_SENDING	DLT_STATE_SENDING	Call of transmit was successful (kDltEventTransmitCallSuccessful)
DLT_STATE_TRY_SENDING	DLT_STATE_READY_TO_SEND	Call of transmit was rejected. (kDltEventTransmitCallRejected)
DLT_STATE_SENDING	DLT_STATE_READY_TO_SEND	Transmission was successful (kDltEventSendingSucceeded)
DLT_STATE_SENDING	DLT_STATE_WAIT_FOR_RECEP	Timeout occurred, transmission has failed. (kDltEventTimeout)

Table 3-3 State machine of DLT with DltCom

If the global state DLT_GLOBAL_STATE_OFFLINE is requested, the sub state machine is left. If the sub state machine is entered again, the initial state is always DLT_STATE_WAIT_FOR_RECEP.

3.4 Main Functions

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

In case of DLT with DltCom the Dlt_MainFunction additionally controls the sub state machine of DLT.

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
0x04	Dlt_SendTraceMessage
0x05	Dlt_RegisterContext
0x0F	Dlt_ComRxIndication
0x10	Dlt_ComTxConfirmation
0x11	Dlt_SetLogLevel
0x12	Dlt_SetTraceStatus
0x13	Dlt_SetVerboseMode
0x50	Dlt_MainFunction
0x51	Dlt_SetState
0x52	Dlt_GetState

Table 3-4 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 3-5 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	■				■				

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_SendLogMessage	■				■	■		■	
Dlt_SendTraceMessage					■	■		■	
Dlt_RegisterContext				■					
Dlt_MainFunction	■							■	■
Dlt_SetState								■	
Dlt_GetState								■	

Table 3-6 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_MAX_NUMBER_OF_DLT_USERS	Defines the number of SWC to use the DLT service. Thus, it defines the number of session IDs.
DLT_IMPLEMENT_VERBOSE_MODE	If defined the functionality for Verbose Mode is available (Precondition: DLT_IMPLEMENT_EXTENDED_HEADER must be defined).
DLT_IMPLEMENT_EXTENDED_HEADER	If defined the extended functionality for the header is available.
DLT_IMPLEMENT_APP_ID_CONTEXT_ID_QUERY	If defined textual description of application ID and context ID are available.
DLT_IMPLEMENT_FILTER_MESSAGES	If defined the functionality for filtering messages is included in the code.
DLT_IMPLEMENT_TIMESTAMP	If defined the timestamp functionality for the header is available.
DLT_MAX_COUNT_APP_IDS	The maximum count of registerable Application Ids.
DLT_MAX_COUNT_CONTEXT_IDS	The maximum count of registerable Context Ids.
DLT_MAX_COUNT_CONTEXT_IDS_PER_APP_ID	This is the maximum count for Context IDs per Application ID.
DLT_MESSAGE_BUFFER_SIZE	Buffer size for storing DLT messages for waiting to transmit over the network (send buffer).
DLT_DEFAULT_MAX_LOG_LEVEL	The maximum log level a received message (from SW-C to Dlt) can have.
DLT_DEFAULT_TRACE_STATUS	If this define is set to STD_ON trace messages are be forwarded by DLT. Else trace messages are rejected.
DLT_FACTORY_DEFAULT_MAX_LOG_LEVEL	The maximum log level a received message (from SW-C to Dlt) can have. Used to set maximum log level to factory default.
DLT_FILTER_MESSAGES	If this definition is set to STD_ON the DLT messages are filtered (Precondition: DLT_IMPLEMENT_FILTER_MESSAGES must be defined).
DLT_ECU_ID	The identifier of the ECU.
DLT_HEADER_PAYLOAD_BYTEORDER	This definition determines the endianness of the CPU (Most Significant Byte).
DLT_HEADER_USE_ECU_ID	If this define is set to STD_ON, the ECU ID is set in standard

Configuration Switch	Description
	header of DLT messages. This define corresponds to field WEID (With ECU ID).
DLT_HEADER_USE_EXTENDED_HEADER	If this define is set to STD_ON, the extended header is attached to DLT messages. This define corresponds to field UEH (Use Extended Header).
DLT_HEADER_USE_SESSION_ID	If this define is set to STD_ON, the Session ID is set in standard header of DLT messages. This define corresponds to field WSID (With Session ID).
DLT_HEADER_USE_TIMESTAMP	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_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.

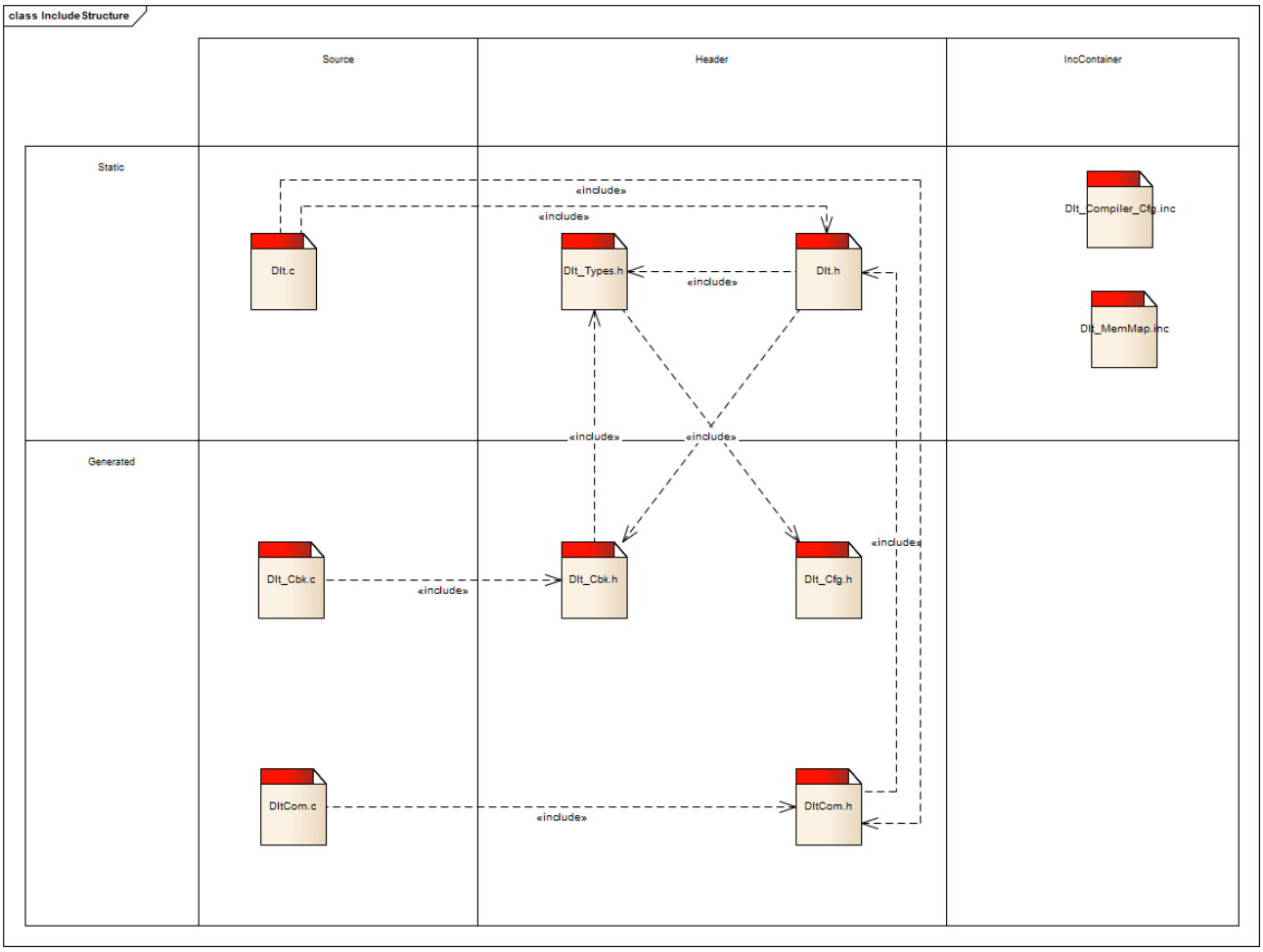


Figure 4-1 Include structure DLT

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:

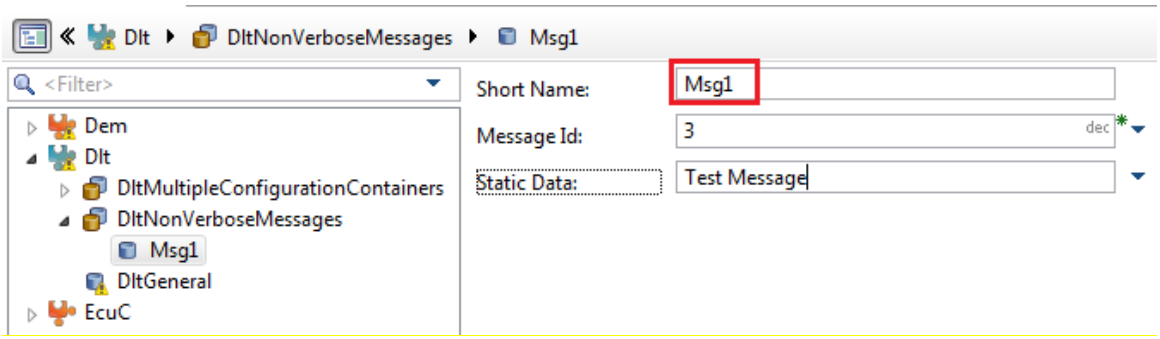


Figure 4-2 Configuration of non-verbose messages

> **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

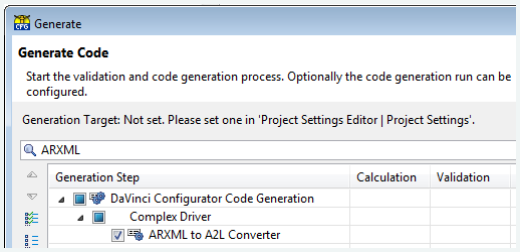
To use any AMD module (DBG, DLT or RTM) with XCP as protocol layer, XCP requires the mapping between symbols and memory addresses. For this purpose there is a master a2l file. This a2l file can then be used by CANoe or CANape to stimulate and/or acquire data from ECU.

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.

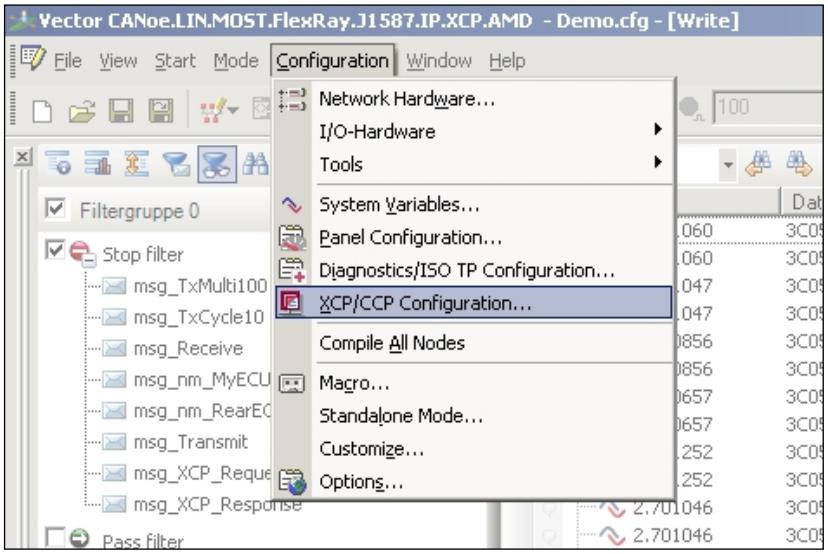


Figure 4-3 Open XCP/CCP option

> 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.

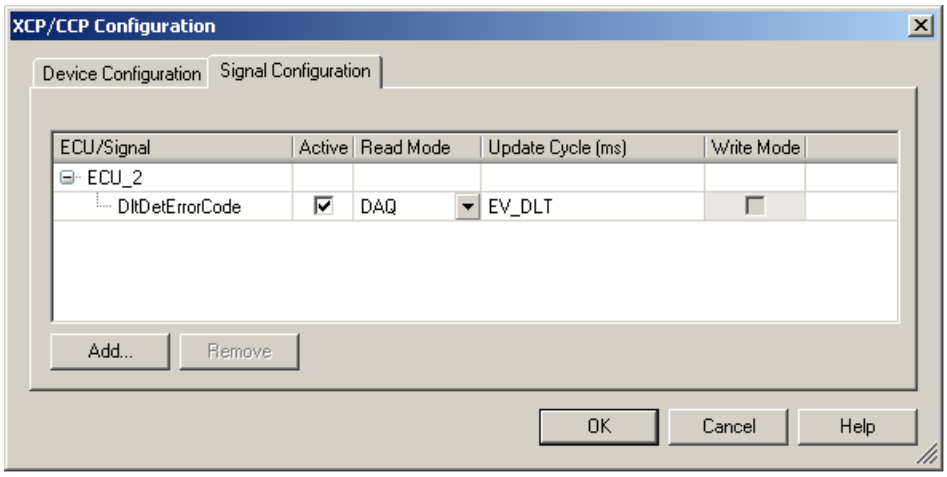


Figure 4-4 Selection of the XCP measurement object for DLT logging of DET reports

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.

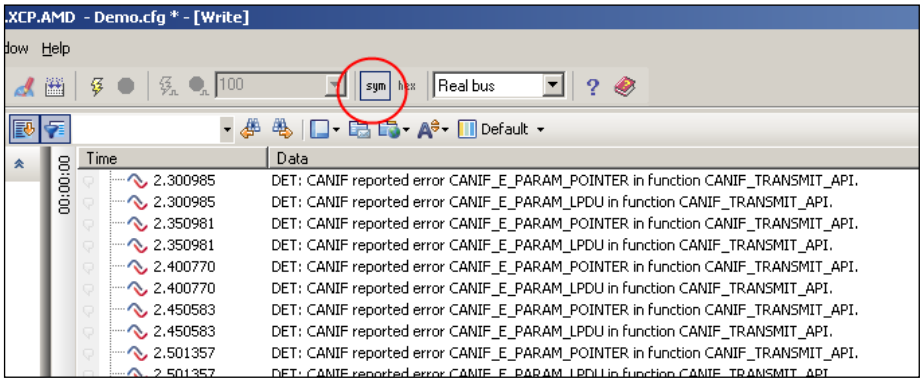


Figure 4-5 Enable the symbolic option by clicking the button [sym] in CANoe's toolbar.

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 -> "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).

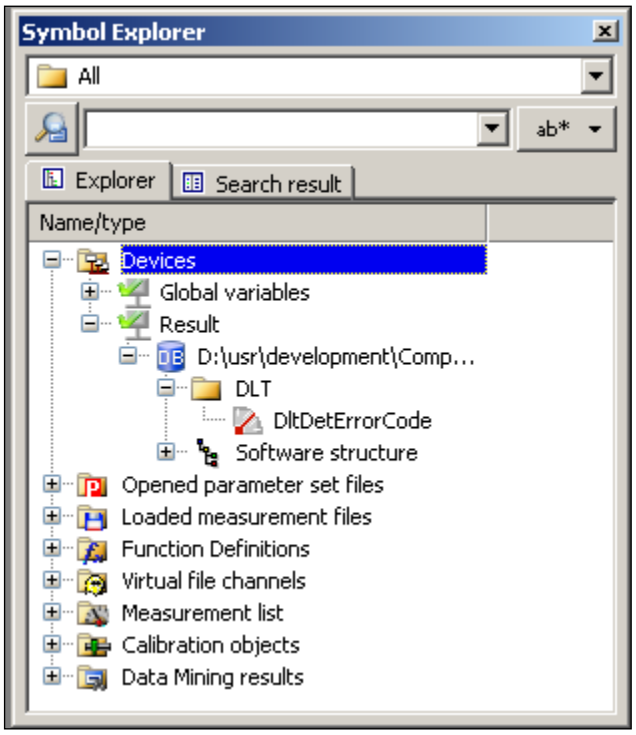


Figure 4-6 CANape's symbol explorer for selection of the DLT variable

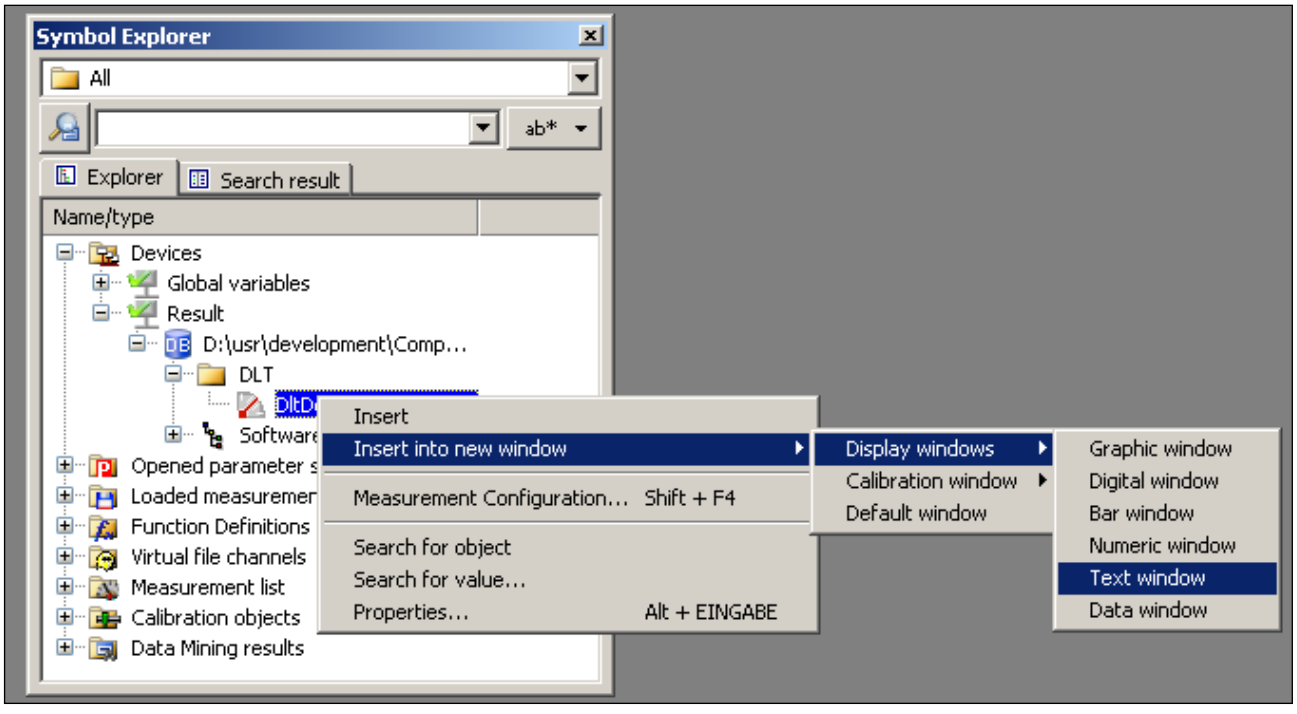


Figure 4-7 Select the DLT variable for measurement in a text window

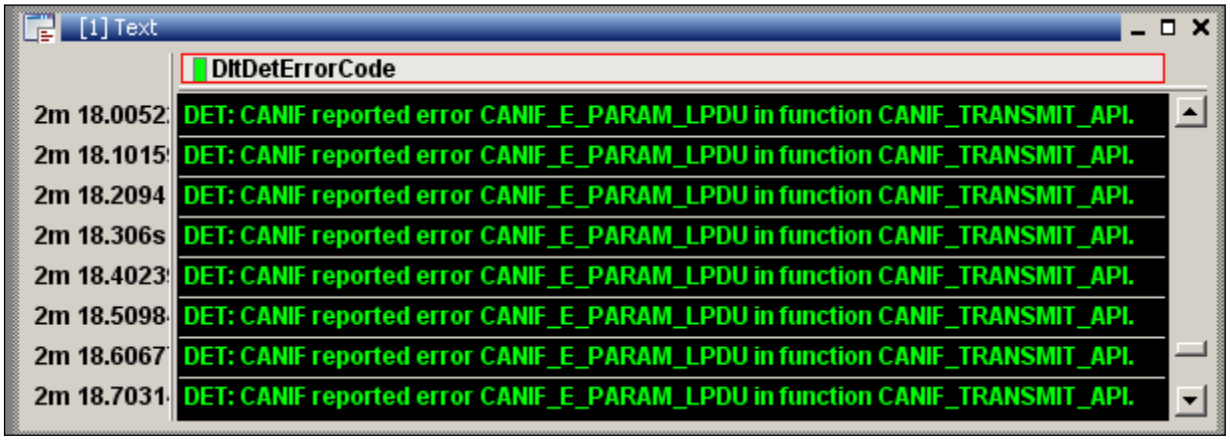


Figure 4-8 Logging of DLT messages using CANape's text window.

4.7 AUTOSAR DLT with com layer

The DLT implemented as specified by AUTOSAR uses a complex driver (CD) as communication layer. This layer is called DltCom. The DltCom is an interface between the upper layer (DLT) and the lower layer. Only SoAd is supported as lower layer.

This feature is licensed and not part of the standard DLT module of Vector.

4.7.1 Configuration

4.7.1.1 DaVinci Configurator Pro

4.7.1.1.1 Complex Driver

The CD "DltCom" can be added in DaVinci Configurator Pro in "Project Settings Editor| Modules", press "Add", choose "Select from Software Integration Package (SIP)" and press "Next". Select "Cdd | /MICROSAR/Cdd_AsrCdd/Cdd" and press "Next". Change "Name" to "DltCom".

Note
The CD has to be named "DltCom" and case sensitivity is required!

In the Basic Editor the DltCom module appears. Open the module and right click on `CddComStackContribution`, choose "Create sub container" -> `CddSoAdUpperLayerContribution`.

The DltCom requires exactly one Tx- and one Rx-PDU. Name the PDUs:

> Pdu_SoAd_DltCom_Rx_IpV4

> Pdu_SoAd_DltCom_Tx_IpV4

Set `CddPduRApiType` to "TP". Furthermore, both CD-PDUs require a reference to global PDUs. The PDUs have to be defined in the EcuC module.

4.7.1.1.2 EcuC

Create two PDUs in `/MICROSAR/EcuC/EcucPduCollection/Pdu`. Call them:

> Pdu_SoAd_DltCom_Rx_IpV4

> Pdu_SoAd_DltCom_Tx_IpV4

Set PduLength to value “512”.

Now the references from DltCom to the PDUs can be added. Therefore open “DltCom” module and add `CddSoAdUpperLayerPduRef` for Tx and Rx.

4.7.1.1.3 SoAd

Now the routing paths in SoAd have to be added.

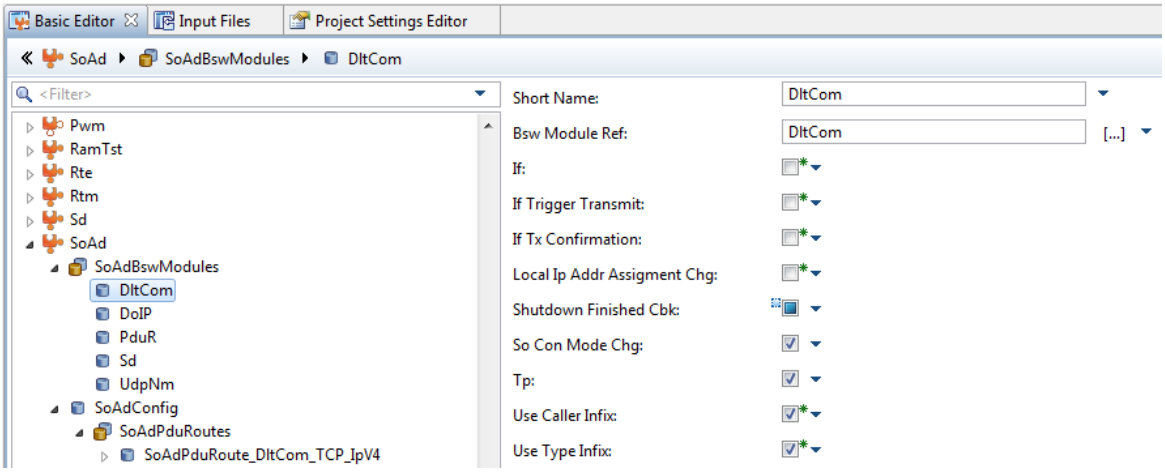
Firstly create a new BSW module in SoAd `/MICROSAR/SoAd/SoAdBswModules`. Name it “DltCom” and select “DltCom” for `SoAdBswModuleRef`.

Activate:

> SoAdSoConModeChg

> SoAdTp

The interface options have to be deactivated.



Socket connection group:

Create a new socket connection group

`/MICROSAR/SoAd/SoAdConfig/SoAdSocketConnectionGroup`.

Name it “SoAdSocketConnectionGroup_DltCom_TCP_IpV4”.

Set active:

> SoAdSocketMsgAcceptanceFilterEnabled

> SoAdSocketSoConModeChgNotification

Set all other true/false options to inactive.

Set `SoAdSocketTpRxBufferMin` to „512“. The TCP port `SoAdSocketLocalPort` has to be set to the required value. For example “13400” can be set in. If “0” is selected, the port is not static and the next free port is used. This is not recommended to be used if the DLT master does need to know the port.

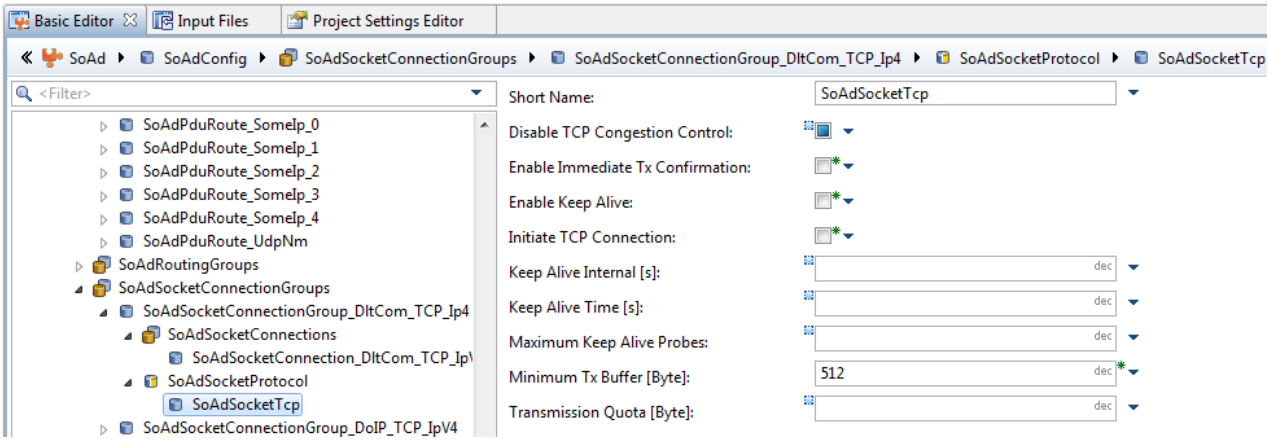
For SoAdSocketLocalAddressRef an address with type „Unicast” and address assignment mode „Fix“ has to be selected.

Name the socket connection SoAdSocketConnection_DltCom_IpV4.

Set SoAdTcpTxQueueSize to „512“.

As socket protocol, TCP has to be chosen

(/MICROSAR/SoAd/SoAdConfig/SoAdSocketConnectionGroup/SoAdSocketProtocol).

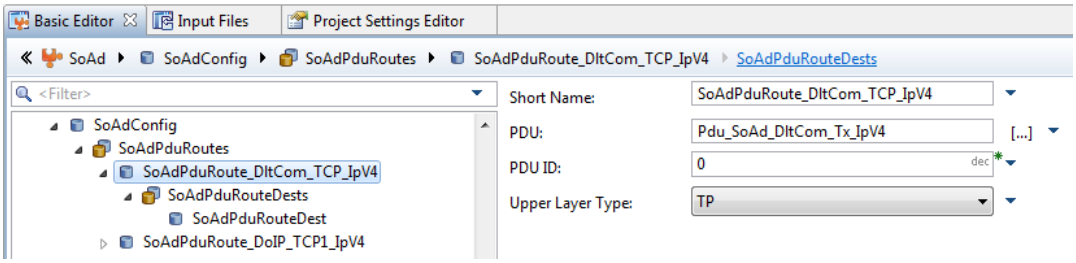


PDU route:

Create a new PDU route (this is the Tx route of the PDU):

Right click on /MICROSAR/SoAd/SoAdConfig/SoAdPduRoute and create new container.

Name the container “SoAdPduRoute_DltCom_TCP_IpV4”, set PDU-reference to “Pdu_SoAd_DltCom_Tx_IpV4” and set SoAdTxUpperLayerType to “TP”.



Afterwards, set

/MICROSAR/SoAd/SoAdConfig/SoAdPduRoute/SoAdPduRouteDest/SoAdTxSocketConnOrSocketConnBundleRef to

“SoAdSocketConnectionGroup_DltCom_TCP_Ip4”.

Socket route:

Now create a socket route (this is the Rx route of the PDU).

Right click on /MICROSAR/SoAd/SoAdConfig/SoAdSocketRoute and create new container. Name it “SoAdSocketRoute_DltCom_TCP_IpV4” and set

/MICROSAR/SoAd/SoAdConfig/SoAdSocketRoute/SoAdRxSocketConnOrSocketConnBundleRef to „SoAdSocketConnectionGroup_DltCom_TCP_Ip4“.

In socket route destination (`/MICROSAR/SoAd/SoAdConfig/SoAdSocketRoute/SoAdSocketRouteDest`) set following parameters:

- SoAdRxPduRef = Pdu_SoAd_DltCom_Rx_IpV4
- SoAdRxPduId = 0
- SoAdRxUpperLayerType = TP

As last step, the DltCom.h has to be included by SoAd because it uses callback functions of DLT. For this purpose the MSR SoAd provides a listing parameter `/MICROSAR/SoAd/SoAdGeneral/SoAdHeaderFileInclusion`.

If this parameter does not exist, the header DltCom.h has to be included by the EthIf with help of a user config file.

4.7.1.1.4 DLT

To use the DLT as specified in AUTOSAR /MICROSAR/Dlt/DltGeneral/DltComLayer has to be set to “DLTCOM”. The “DltCom” requires the existence of container `/MICROSAR/Dlt/DltMemory` and `/MICROSAR/Dlt/DltMultipleConfigurationContainer/DltMessageFiltering`. Additionally the DltEcuId has to exist.

If the license is valid and the configuration is done as described above, no validation error should occur. Else check the configuration.

4.7.1.2 DaVinci Developer

The DLT provides services that can be used by the application. If a SWC wants to use these services, it has to provide other services itself.

Services provided by DLT:

- Dlt_RegisterContext
- Dlt_SendLogMessage
- Dlt_SendTraceMessge

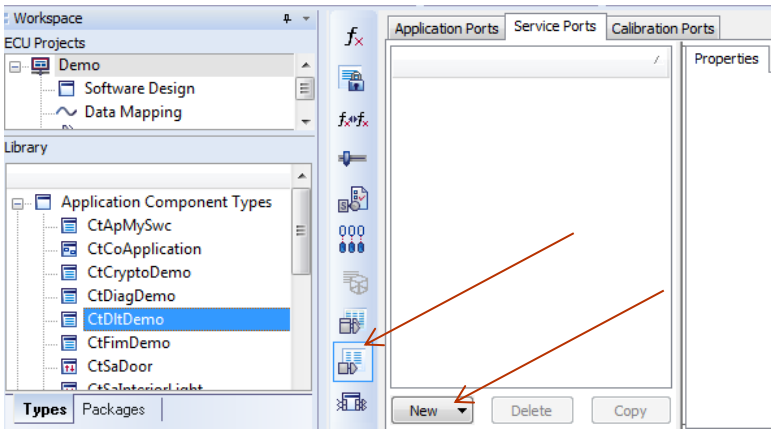
Services provided by SWC:

- SetLogLevel
- SetTraceStatus
- SetVerboseMode

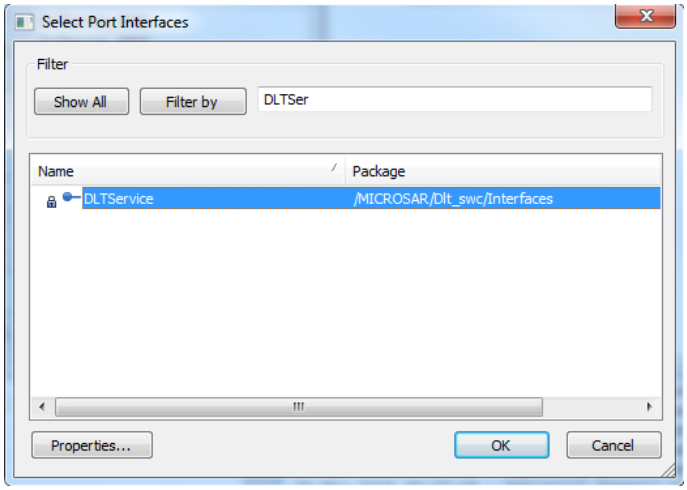
All services have to be mapped to runnables of the SWC.

For example, you have a SWC named CtDltDemo and two DLT users. At first you have to add “Service Ports”, these ports contain all required services. Open the SWC CtDltDemo with double click and select “Port Prototype List”.

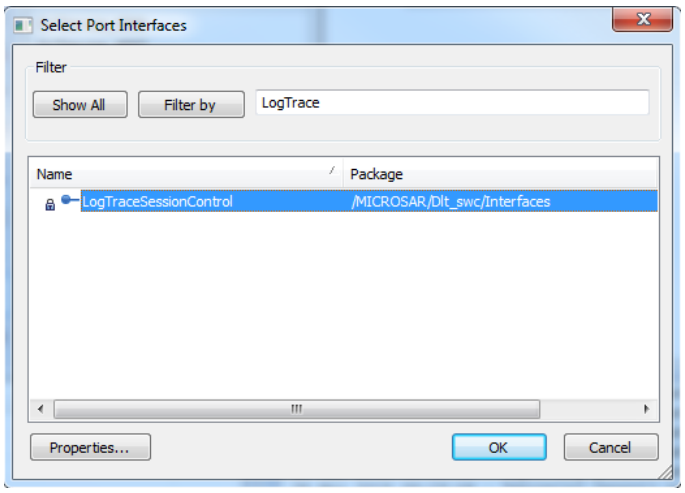
Because we have two users, we require four ports. Each user gets one server port and one client port. To add these ports press “New” -> “From Port Interface”.



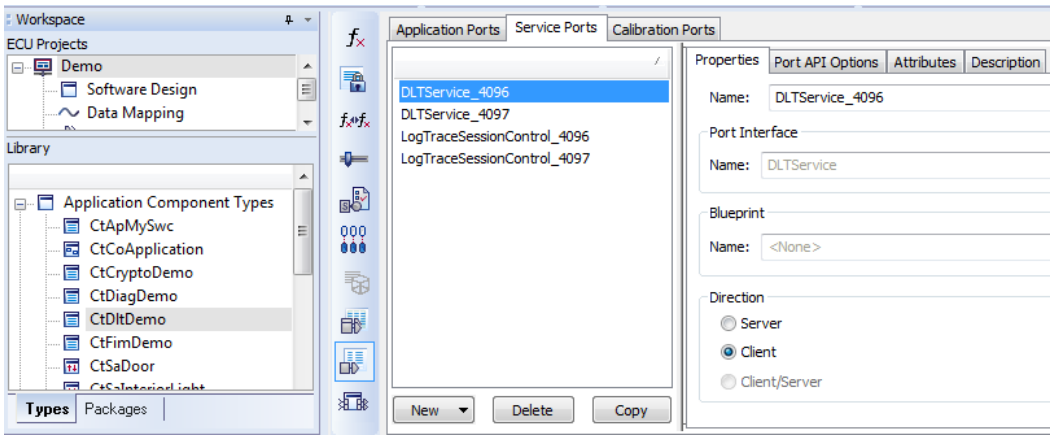
The “Select Port Interface” window opens. Add DLTService two times.



Afterwards add “LogTraceSessionControl” two times.



Now open the properties of each port and change their names. It is recommended to use their session id within their names. The session ids start at 4096 and should be continuous. The DLTService ports are provided by DLT, thus they are Client ports. The LogTraceSessionControl have to be provided by the SWC, thus they are server ports.

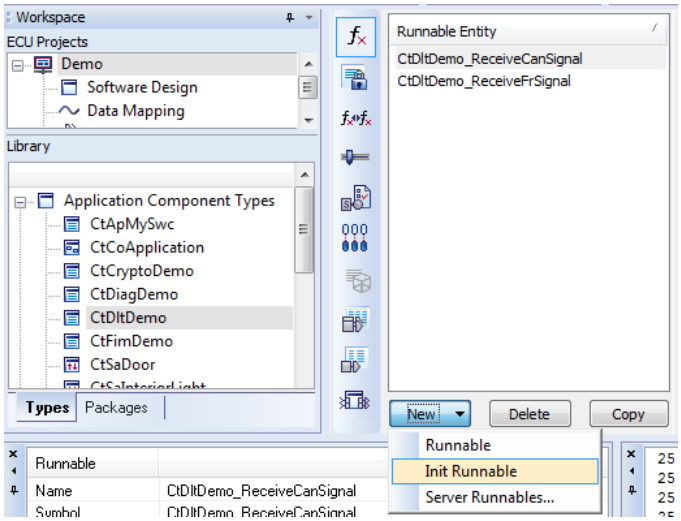


Now the SWC requires the runnables, which are triggered by DLT and the runnables calling the DLT services.

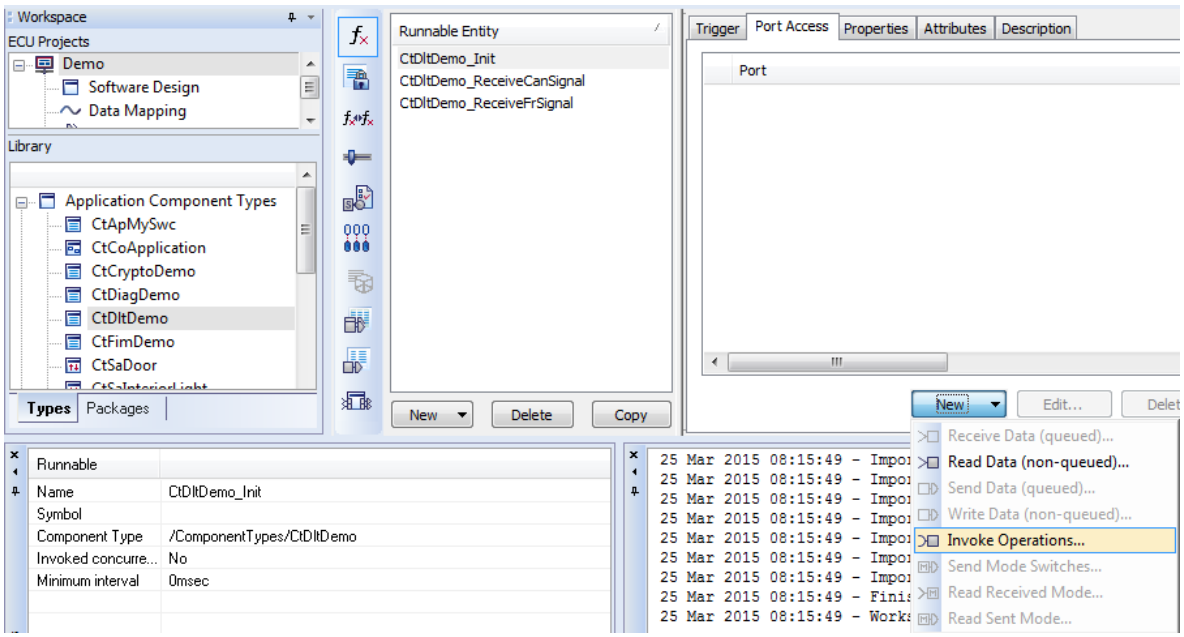
Before the SWC can use the DLT services, it has to register its contexts. The context registration has to be done once for each context. Thus, it is recommended to use an init-runnable to register all contexts of the SWC.

Note
It is possible to register multiple contexts within one session.

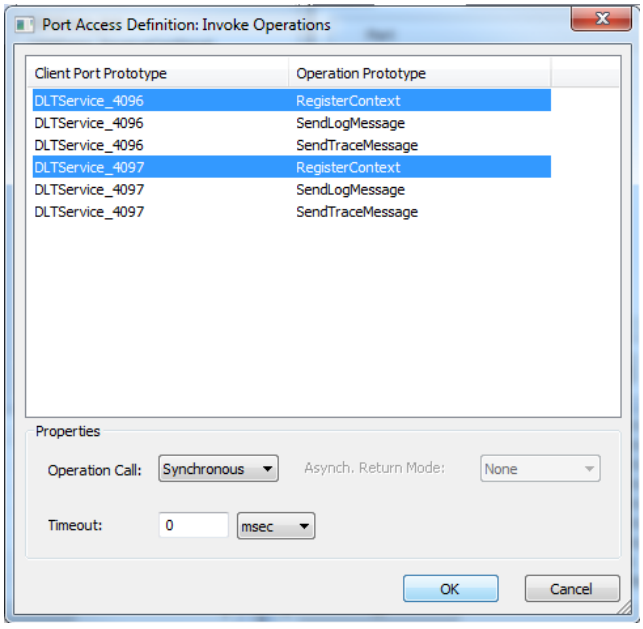
To add an init-runnable open “Runnable Entity List”, press “New” and select “Init Runnable”.



This runnable should call the context registration services, thus open “Port Access” and press “New” -> “Invoke Operations…”.



The “Port Access Definition: Invoke Operations” window opens. Select all “RegisterContext” operations and press “OK”.



In runnables that are called cyclically or event triggered, the services “SendLogLevel” and/or “SendTraceStatus” can be called. They are added same way as the “RegisterContext“. For example, the CtDltDemo_ReceiveCanSignal calls SendLogMessage and CtDltDemo_ReceiveFrSignal calls SendTraceMessage.

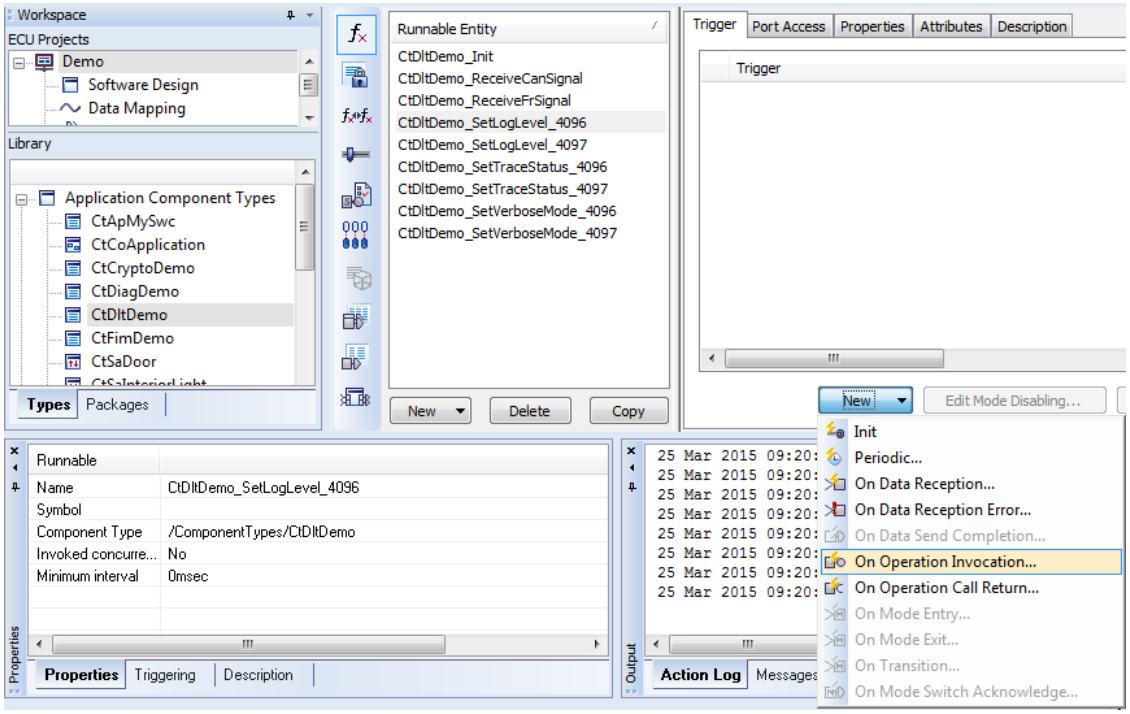
Now the runnables, triggered by DLT have to be added. Add six runnables, rename them and set triggers:

Runnable name	Trigger
CtDltDemo_SetLogLevel_4096	LogTraceSessionControl_4096.SetLogLevel
CtDltDemo_SetTraceStatus_4096	LogTraceSessionControl_4096.SetTraceStatus
CtDltDemo_SetVerboseMode_4096	LogTraceSessionControl_4096.SetVerboseMode

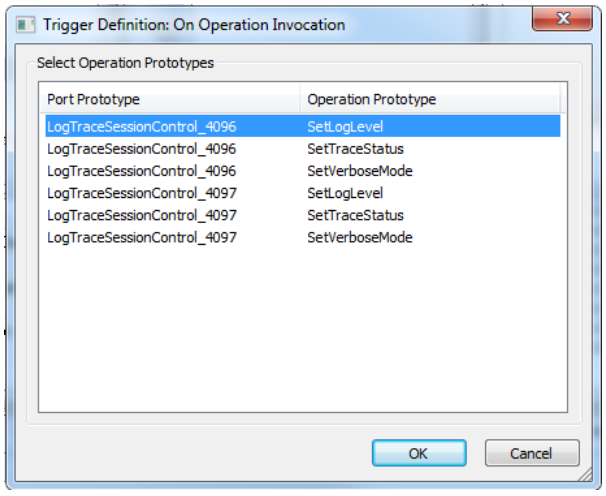
Runnable name	Trigger
CtDltDemo_SetLogLevel_4097	LogTraceSessionControl_4096.SetLogLevel
CtDltDemo_SetTraceStatus_4097	LogTraceSessionControl_4096.SetTraceStatus
CtDltDemo_SetVerboseMode_4097	LogTraceSessionControl_4096.SetVerboseMode

Table 4-7 Client runnables of SWC (example names and assigned triggers)

The trigger can be added in “Trigger” view by pressing “New” -> “On Operation Invocation”.



The “Trigger Definition: On Operation Invocation” window opens. Select the fitting port and operation prototype and press “OK”.



Save the changes and close the DaVinci Developer.

DaVinci Configurator Pro:

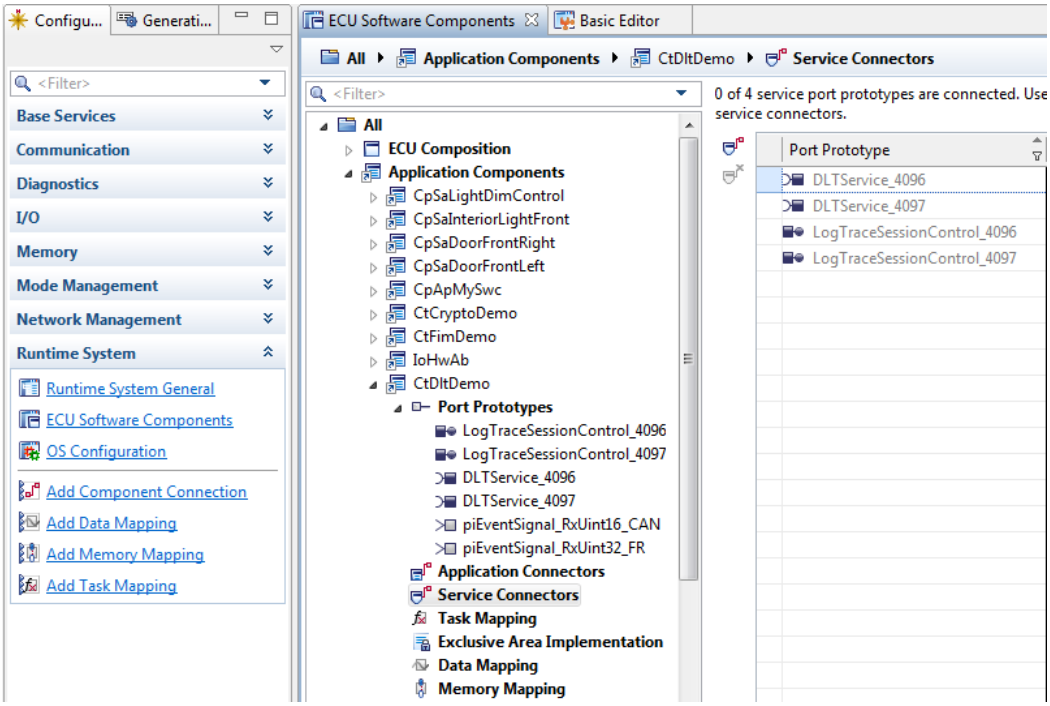
Now you have to open the DaVinci Configurator Pro again. Now a Validation error of RTE (RTE01056) should occur because of unmapped runnable entities.

RTE01056	Unmapped runnable entity (1 message)
RTE01056	The runnable entity <CtDltDemo_Init> is not mapped to a valid task.
	/ActiveEcuC/Rte/CtDltDemo_EcuSwComposition/CtDltDemo_InitEvent
	RteMappedToTaskRef

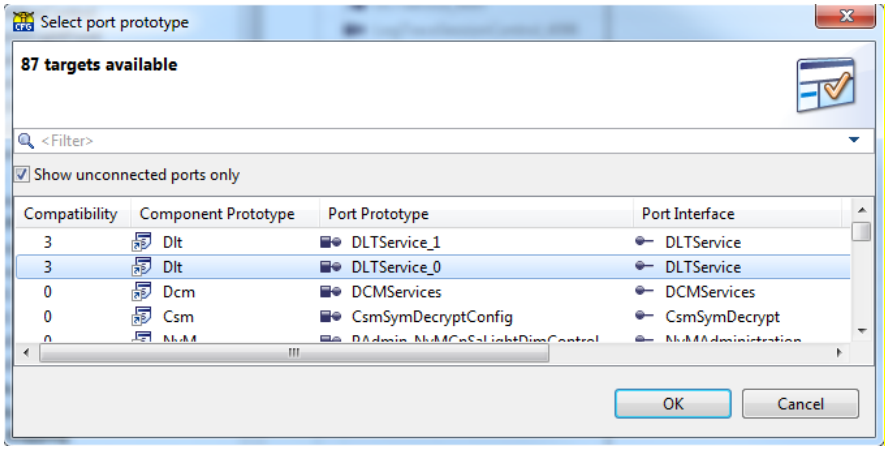
Double click on “/ActiveEcuC/Rte/CtDltDemo_EcuSwComposition/CtDltDemo_InitEvent”. Now map the init-runnable to an init-task and add a valid position in the task (RtePositionInTask). The error should disappear.

Now the ports of the SWC (CtDltDemo) have to be mapped to the ports of DLT. Therefore expand “Runtime System” in the “Configuration Editors” and select “ECU Software Components”.

Expand “All” -> “Application Components” -> “CtDltDemo” and select “Service Connectors”.



Now select “Connect” for each port and choose the corresponding port of DLT.

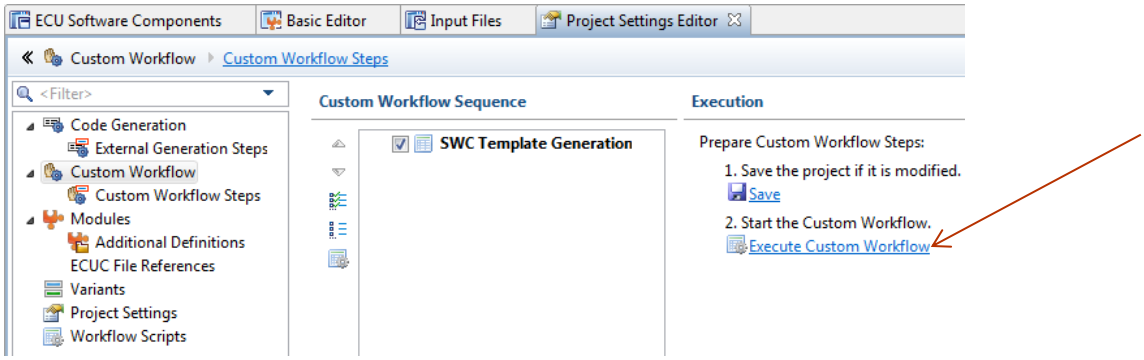


It is recommended to map “DLTService_4096” to “DLTService_0” and “DLTService_4097” to “DLTService_1” and so on because the value 4096 is the offset for the sessions.

The same goes for the “LogTraceSessionControl”, map “LogTraceSessionControl_4096” to “PSCN000” and “LogTraceSessionControl_4097” to “PSCN001”.

Port Prototype	Connected Co	Connected Port Prototype
DLTService_4096	Dlt	DLTService_0
DLTService_4097	Dlt	DLTService_1
LogTraceSessionControl_4096	Dlt	PSCN000
LogTraceSessionControl_4097	Dlt	PSCN001

To update the implementation template of your SWC open the “Project Settings Editor” and check if the “Custom Workflow” contains a Custom Workflow Step with the parameter “Rte: -g I”. Select “Custom Workflow” and select “Execute Custom Workflow”.



Now the configuration for DLT on AUTOSAR is done and the embedded code has to be adapted.

4.7.2 Information about DLTs’ IDs

DLT requires a lot of different IDs. Thus, here is some information about these IDs:

The couple of application ID and context ID define one message of one DLT user. Multiple messages can be part of one session. Thus, in one session each couple of application ID and context ID has to be unique. Because one application ID can contain multiple context IDs, each context ID must be unique in the scope of one application ID.

DLT supports two transmission modes:

- Verbose mode
- Non-verbose mode

In case of non-verbose mode, less data is transmitted to the DLT master (i.e. the ‘static data’ which is always constant is left away). For information about the DLT master, refer to chapter 4.7.7). This (‘static’) data is required for the DLT master to know. Thus the DLT master requires another source for it. This source is a FIBEX file. To refer to the FIBEX-file message IDs are used. These message IDs have to be transmitted within a non-verbose message and have to be described in the FIBEX-file. Each DLT user requires one message ID.

Because multiple ECUs in one automobile should be able to use the DLT simultaneously, there is ECU IDs. These ECU IDs are unique in one automobile.

ID	relation	ID
Application ID	1:n	Context ID

ID	relation	ID
Session ID	1:n	Application ID | Context ID
Message ID	1:1	Application ID | Context ID
DLT user	1:1	Session ID
ECU ID	1:n	Session ID

Table 4-8 Relations of IDs

4.7.3 DLT protocol

The DLT protocol is implemented as specified in chapter 7.7 of AUTOSAR DLT specification [1].

Length (bytes)	Name	Description
4,8,12 or 16	Standard Header (mandatory)	Contains essential information for interpreting the DLT message.
10	Extended Header (optional)	Can be added optionally for providing more information or for use in control messages.
x	Payload (optional)	Contains information about a specific log and trace message.

Table 4-9 DLT protocol as specified in [1]

In the standard header it is defined which information the header contains and how long the message is.

Caution
The bit MSBF (most significant bit first) in standard header only defines the endianness of payload. The headers (standard and extended) are always in big endian as specified by [Dlt091] in [1].

4.7.3.1.1 Verbose mode

If the SWC wants to send a log or trace message, it can be send in verbose or in non-verbose mode.

If set to verbose mode, the message should send all dynamic and static data otherwise the static data should be replaced by the corresponding message ID. The message ID has to be send at first place in the payload.

In verbose mode the payload contains one or more couples of static and dynamic data. The static data is always 32bit and describes the following dynamic value. For more information about static and dynamic data in log and trace messages refer to [1].

4.7.4 Message types

There are three kinds of messages.

4.7.4.1 Log messages

Log messages are always sent from application via DLT to a DLT master.

The size of a log message can be between 8 and DltMaxMessageLength bytes. It always contains the standard header. In non-verbose mode the 4 byte message ID is added. In verbose mode the external header is added. Additionally the optional payload is added.

4.7.4.2 Trace messages

Trace messages are always sent from application or RTE via DLT to a DLT master.

The size of a trace message can be between 8 and DltMaxMessageLength bytes. It always contains the standard header. In non-verbose mode the 4 byte message ID is added. In verbose mode the external header is added. Additionally the optional payload is added.

4.7.4.3 Control messages

Control messages are always sent between DLT and DLT master. They are used to configure the registered contexts of DLT users. Furthermore they are used by DLT master to request context information.

Control messages always contain standard header and a service ID. Additional payload is optional and depends on the service, whereas the extended header is never used for control messages. Therefore refer to chapter 7.7.7.1 of AUTOSAR DLT specification [1].

If the requested service is not supported, it returns a control message with one parameter matching “not supported”.

4.7.5 Message Filtering

4.7.5.1.1 Log level filter

To reduce the number of DLT messages, it is possible to filter messages which are currently not important.

The filtering can be done by the DLT if DltImplementFilterMessages is set to true. Furthermore the parameter DltFilterMessages has to be true or the service SetMessageFilterering is called to activate the filtering.

The disadvantage is that the log message has to be copied to the DLT first before it can be rejected.

Therefore it is possible to filter the DLT message already in the application. For this purpose the global variable Dlt_DefaultMaxLogLevel has to be compared with the log level of the DLT user. If the log level is higher than the maximal log level, the message must be rejected.

To be able to access the variable Dlt_DefaultMaxLogLevel it is required to include Dlt.h in the application.

4.7.5.1.2 Trace status

If the application wants to send a trace message, the trace status of the DLT user has to be checked. If the status is “turned off” the message should be rejected else it can be send.

The filtering can be done by the DLT if DltImplementFilterMessages is set to true. Furthermore the parameter DltFilterMessages has to be true or the service SetMessageFilterering is called to activate the filtering.

The disadvantage is that the log message has to be copied to the DLT first before it can be rejected.

Therefore it is possible to filter the DLT message already in the application. For this purpose the global variable Dlt_DefaultTraceStatus has to be compared with the log level of the DLT user. If the log level is higher than the maximal log level, the message must be rejected.

To be able to access the variable Dlt_DefaultTraceStatus it is required to include Dlt.h in the application.

4.7.6 Sending log and trace messages from SWC

If you have done all steps of configuration described in chapter 4.7.1, you can add the embedded part of communication between DLT and SWCs. Here the same example is used.

In Table 4-10 all information of the example are listed. There is one SWC (CtDltDemo), two runnables each handling 2 messages with different log levels and verbose modes.

Runnable	Message ID	Application ID	Context ID	Session ID	Log Level	Verbose Mode
CtDltDemo_ReceiveCanSignal	3	“CtD1” = 0x43744431	“ReC1” = 0x52654331	4096	Warn	Non-Verb
CtDltDemo_ReceiveCanSignal	4	“CtD1” = 0x43744431	“ReC2” = 0x52654332	4096	Fatal	Verb
CtDltDemo_ReceiveFrSignal	5	“CtD2” = 0x43744432	“ReF1” = 0x52654631	4097	Error	Verb
CtDltDemo_ReceiveFrSignal	6	“CtD2” = 0x43744432	“ReF2” = 0x52654632	4097	Verb	Non-Verb

Table 4-10 Example configuration of all DLT user messages

In this example the default threshold for log messages is “DLT_LOG_INFO”.

To reach this configuration some adaptions have to be done in DaVinci Configurator Pro and DaVinci Developer:

Configuration switch	state
/MICROSAR/Dlt/DltGeneral/DltImplementVerboseMode	Active
/MICROSAR/Dlt/DltGeneral/DltImplementFilterMessages	Active
/MICROSAR/Dlt/DltNonVerboseMessage	> 4
/MICROSAR/Dlt/DltMemory/DltMaxCountContextIds	> 4
/MICROSAR/Dlt/DltMemory/DltMaxCountContextIdsPerAppId	> 2
/MICROSAR/Dlt/DltMultipleConfigurationContainer/DltMessageFiltering/DltDefaultMaxLogLevel	4 (= INFO)
/MICROSAR/Dlt/DltMultipleConfigurationContainer/DltProtocol/DltUseVerboseMode	Active
/MICROSAR/Dlt/DltNonVerboseMessage /DltNonVerboseMessage_CanRx_1 -> Message Id = 3 /DltNonVerboseMessage_CanRx_2 -> Message Id = 4 /DltNonVerboseMessage_FrRx_1 -> Message Id = 5 /DltNonVerboseMessage_FrRx_2 -> Message Id = 6	

Table 4-11 Example configuration in DaVinci Configurator Pro

In DaVinci Developer the runnable “CtDltDemo_ReceiveCanSignal” should access the port DLTService_4096.SendLogMessage. The runnable “CtDltDemo_ReceiveFrSignal” should access DLTService_4097.SendLogMessage.

Generate the SWCs and the DLT.

4.7.6.1 Context Registration

Before you can use the DLT services, the SWC has to register its context/s. In the example before, we added an init-runnable to the SWC and named it CtDltDemo_Init. Within this runnable the contexts can be registered by calling following functions:

- Rte_Call_DLTService_4096_RegisterContext
- Rte_Call_DLTService_4097_RegisterContext
- …

Each API should be called for each message of one session ID.

You have to pass an application ID and a context ID (all other parameter can be 0 respectively NULL_PTR). For the example, call both APIs twice to register all four couples of application and context IDs, according Table 4-10.

A possible implementation is shown in the following example.

Example

```
#include "Rte_CtDltDemo.h"
#include "Dlt.h"
…

Dlt_AppIDsType CtDltDemo_DltContext[2]; /* Array size == number of used session IDs */
…

FUNC(void, CtDltDemo CODE) CtDltDemo Init(void)
{
  Dlt_ReturnType retVal;

  CtDltDemo_DltContext[0].app_id = 0x43744431; /* CtD1 */
  CtDltDemo DltContext[0].app description = NULL PTR;
  CtDltDemo DltContext[0].len app description = 0;
  CtDltDemo DltContext[0].count context ids = 2;

  CtDltDemo_DltContext[0].context_id_info[0].context_id = 0x52654331; /* ReC1 */
  CtDltDemo_DltContext[0].context_id_info[0].log_level = 0; /* Will be set by DLT */
  CtDltDemo DltContext[0].context id info[0].trace status = 0; /* Will be set by DLT */
  CtDltDemo DltContext[0].context id info[0].len context description = 0;
  CtDltDemo_DltContext[0].context_id_info[0].context_description = NULL_PTR;
  CtDltDemo_DltContext[0].context_id_info[0].cbk_info.verbose_mode = 0; /* Will be set
by DLT */

  CtDltDemo DltContext[0].context id info[1].context id = 0x52654332; /* ReC2 */
  CtDltDemo_DltContext[0].context_id_info[1].log_level = 0; /* Will be set by DLT */
  CtDltDemo_DltContext[0].context_id_info[1].trace_status = 0; /* Will be set by DLT */
  CtDltDemo_DltContext[0].context_id_info[1].len_context_description = 0;
  CtDltDemo_DltContext[0].context_id_info[1].context_description = NULL_PTR;
  CtDltDemo DltContext[0].context id info[1].cbk info.verbose mode = 0; /* Will be set
by DLT */

  CtDltDemo_DltContext[1].app_id = 0x43744432; /* CtD1 */
  CtDltDemo_DltContext[1].app_description = NULL_PTR;
  CtDltDemo_DltContext[1].len_app_description = 0;
  CtDltDemo DltContext[1].count context ids = 2;

  CtDltDemo_DltContext[1].context_id_info[0].context_id = 0x52654631; /* ReF1 */
  CtDltDemo_DltContext[1].context_id_info[0].log_level = 0; /* Will be set by DLT */
  CtDltDemo_DltContext[1].context_id_info[0].trace_status = 0; /* Will be set by DLT */
  CtDltDemo DltContext[1].context id info[0].len context description = 0;
  CtDltDemo DltContext[1].context id info[0].context description = NULL PTR;
  CtDltDemo DltContext[1].context id info[0].cbk info.verbose mode = 0; /* Will be set
by DLT */

  CtDltDemo_DltContext[1].context_id_info[1].context_id = 0x52654632; /* ReF2 */
  CtDltDemo_DltContext[1].context_id_info[1].log_level = 0; /* Will be set by DLT */
  CtDltDemo DltContext[1].context id info[1].trace status = 0; /* Will be set by DLT */
  CtDltDemo_DltContext[1].context_id_info[1].len_context_description = 0;
  CtDltDemo_DltContext[1].context_id_info[1].context_description = NULL_PTR;
  CtDltDemo_DltContext[1].context_id_info[1].cbk_info.verbose_mode = 0; /* Will be set
by DLT */

  retVal = (Dlt ReturnType)Rte Call DLTService 4096 RegisterContext(
                           CtDltDemo_DltContext[0].app_id,
                           CtDltDemo_DltContext[0].context_id_info[0].context_id,
                           NULL_PTR, 0, NULL_PTR, 0);

  /* Check return value */

  retVal = (Dlt_ReturnType)Rte_Call_DLTService_4096_RegisterContext(
                           CtDltDemo_DltContext[0].app_id,
                           CtDltDemo DltContext[0].context id info[1].context id,
                           NULL PTR, 0, NULL PTR, 0);

  /* Check return value */

  retVal = (Dlt_ReturnType)Rte_Call_DLTService_4097_RegisterContext(
                           CtDltDemo DltContext[1].app id,
                           CtDltDemo DltContext[1].context id info[0].context id,
                           NULL PTR, 0, NULL PTR, 0);
```

```
  /* Check return value */

  retVal = (Dlt_ReturnType)Rte_Call_DLTService_4097_RegisterContext(
                             CtDltDemo_DltContext[1].app_id,
                             CtDltDemo_DltContext[1].context_id_info[1].context_id,
                             NULL_PTR, 0, NULL_PTR, 0);
  /* Check return value */
}
```

Note
Only the contexts of VFB traces and SWCs have to be registered. The contexts of DEM and DET events are initially registered and can be used from ECU start up on.

For each succeeded registration DLT calls the following APIs of SWC (here: CtDltDemo):

- CtDltDemo_SetLogLevel_<XYZ>
- CtDltDemo_SetTraceStatus_<XYZ>
- CtDltDemo_SetVerboseMode_<XYZ>

Where <XYZ> stands for the session ID. With a call to these APIs the SWC is informed about the log level, trace status and the verbose mode it should be using. The APIs can also be called after initialization, triggered by the DLT master. All APIs get an application ID and a context ID, with these IDs the SWC knows which message context has to be changed.

Example

```
FUNC(Std_ReturnType, CtDltDemo_CODE) CtDltDemo_SetLogLevel_4096(Dlt_ApplicationIDType
        AppId, Dlt_ContextIDType ContextId, Dlt_MessageLogLevelType LogLevel)
{
  Std_ReturnType retVal = E_NOT_OK;

  if (CtDltDemo_DltContext[0].app_id == AppId)
  {
    if (CtDltDemo_DltContext[0].context_id_info[0].context_id == ContextId)
    {
      CtDltDemo_DltContext[0].context_id_info[0].log_level = (sint8)LogLevel;
      retVal = E_OK;
    }
    else if (CtDltDemo_DltContext[0].context_id_info[1].context_id == ContextId)
    {
      CtDltDemo_DltContext[0].context_id_info[1].log_level = (sint8)LogLevel;
      retVal = E_OK;
    }
  }

  return retVal;
}
/* Do the same thing for CtDltDemo SetTraceStatus 4096/4097,
CtDltDemo_SetVerboseMode_4096/4097 and CtDltDemo_SetLogLevel_4097 */
```

Now all messages have the default value for log level, trace status and verbose mode. Thus the values depend on your configuration in DaVinci Configurator Pro:

- /MICROSAR/Dlt/DltMultipleConfigurationContainer/DltMessageFiltering/DltDefaultMaxLogLevel
- /MICROSAR/Dlt/DltMultipleConfigurationContainer/DltMessageFiltering/DltDefaultTraceStatus
- /MICROSAR/Dlt/DltMultipleConfigurationContainer/DltProtocol/DltUseVerboseMode

To change these values to the required, the DLT master (e.g. DltViewer) has to trigger the services:

- Set_LogLevel
- Set_TraceStatus
- SetVerboseMode

Therefore, each couple of Application ID and Context ID has to be set individually or globally. In the example in Table 4-10, each value has to be change individually because they differ in value.

Caution
With the DltViewer it is not possible to set the verbose mode individually.

4.7.6.2 Sending of log and trace messages

As soon as the context is registered, log and trace messages can be send. Therefore the function like macros can be called in context of the configured runnables:

- Rte_Call_DLTService_4096_SendLogMessage
- Rte_Call_DLTService_4097_SendLogMessage
- Rte_Call_DLTService_4096_SendTraceMessage
- Rte_Call_DLTService_4097_SendTraceMessage
- …

The log message service requires log info, log data and log data length. The parameters are explained in chapter 5.2.7, but session ID is not part of the signature. The session ID is set by the RTE.

The trace message service requires trace info, trace data and trace data length. The parameters are explained in chapter 5.2.8, but session ID is not part of the signature. The session ID is set by the RTE.

Example

```
FUNC(void, CtDltDemo_CODE) CtDltDemo_ReceiveCanSignal(void)
{
  Dlt_ReturnType retVal;
  Dlt_MessageLogInfoType logInfo; /* arg1 */
  uint8 logData[DLT_MAX_MESSAGE_LENGTH];
  uint16 logDataIndex = 0;

  /* Optional: Only required if you want to check if the message is allowed to be send
               in verbose mode. */
  boolean useExtendedHeader = FALSE;
  boolean useVerboseMode    = FALSE;

  /* Dynamic data for payload */
  uint16 logVar1;
  uint8  logVar2;

  /*
   *  Send first log message in non-verbose mode
   */
  logInfo.arg_count =  2; /* There are 2 log variables (logVar1 and logVar2) */
  logInfo.app_id =     CtDltDemo_DltContext[0].app_id;
  logInfo.context_id = CtDltDemo_DltContext[0].context_id_info[0].context_id;
  logInfo.options =    DLT_NON_VERBOSE_MSG | DLT_TYPE_LOG;
  logDataIndex =       7; /* Message ID (4Byte) + logVar1 (2Byte) + logvar2 (1Byte) */
  logVar1 =            0xFFFF;
  logVar2 =            0x01;

  /* Optional: Check if log level in pass through range. Check can be omitted if
   /MICROSAR/Dlt/DltMultipleConfigurationContainer/DltMessageFiltering/DltFilterMessages
     is active.
   */
  if (CtDltDemo_DltContext[0].context_id_info[0].log_level <=
      (sint8)Dlt_DefaultMaxLogLevel)
  {
    /* Check endianness */
#if defined (DLT_HEADER_PAYLOAD_BYTEORDER) && (DLT_HEADER_PAYLOAD_BYTEORDER ==
DLT_BIGENDIAN)
    /* Set message ID as first parameter in payload */
    logData[0] =
(uint8)((uint32)DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1 >> 24);
    logData[1] =
(uint8)((uint32)DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1 >> 16);
    logData[2] =
(uint8)((uint32)DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1 >>  8);
    logData[3] = (uint8)(DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1);
    logData[4] = (uint8)(logVar1 >> 8); /* First log variable */
    logData[5] = (uint8)(logVar1);      /* First log variable */
    logData[6] = (uint8)(logVar2); /* Second log variable */
#else
    /* Set message ID as first parameter in payload */
    logData[0] = (uint8)(DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1);
    logData[1] =
```

```
(uint8)((uint32)DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1 >>  8);
    logData[2] =
(uint8)((uint32)DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1 >> 16);
    logData[3] =
(uint8)((uint32)DltConf_DltNonVerboseMessage_DltNonVerboseMessage_CanRx_1 >> 24);
    logData[4] = (uint8)(logVar1);      /* First log variable */
    logData[5] = (uint8)(logVar1 >> 8); /* First log variable */
    logData[6] = (uint8)(logVar2); /* Second log variable */
#endif

    retVal = Rte_Call_DLTService_4096_SendLogMessage( &logInfo, logData, logDataIndex);
    /* Check return value… */
  }

  /*
   *  Send second log message in verbose mode
   */
  logInfo.arg_count =  2; /* There are 2 log variables (logVar1 and logVar2) */
  logInfo.app_id =     CtDltDemo_DltContext[0].app_id;
  logInfo.context_id = CtDltDemo_DltContext[0].context_id_info[1].context_id;
  logInfo.options = DLT_VERBOSE_MSG | DLT_TYPE_LOG;
  logDataIndex = 11; /* Type Info (4Byte) + logVar1 (2Byte) + Type info (4Byte) +
                        logvar2 (1Byte) */
  logVar1 = 0x0001;
  logVar2 = 0xFF;

  /* Optional: Check if verbose mode active */
#if defined (DLT_IMPLEMENT_EXTENDED_HEADER)
  useExtendedHeader = Dlt_HeaderUseExtendedHeader;
#endif /* (DLT_IMPLEMENT_EXTENDED_HEADER) */
#if defined (DLT_IMPLEMENT_VERBOSE_MODE)
  useVerboseMode = Dlt_HeaderUseVerboseMode;
#endif /* (DLT_IMPLEMENT_VERBOSE_MODE) */

  if ((useVerboseMode == TRUE) && (useExtendedHeader == TRUE))
  {
    /* Optional check if log level in pass through range. Can be omitted if
   /MICROSAR/Dlt/DltMultipleConfigurationContainer/DltMessageFiltering/DltFilterMessages
       is active.
     */
    if (CtDltDemo_DltContext[0].context_id_info[1].log_level <=
        (sint8)Dlt_DefaultMaxLogLevel)
    {
      /* Check endianness */
#if defined (DLT_HEADER_PAYLOAD_BYTEORDER) && (DLT_HEADER_PAYLOAD_BYTEORDER ==
DLT_BIGENDIAN)
      logData[ 0] = 0;    /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 1] = 0;    /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 2] = 0;    /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 3] = 0x42; /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 4] = (uint8)(logVar1 >> 8); /* First log variable */
      logData[ 5] = (uint8)(logVar1);      /* First log variable */
      logData[ 6] = 0;    /* Type Info - logVar2 - Unsigned int, 8 bit */
      logData[ 7] = 0;    /* Type Info - logVar2 - Unsigned int, 8 bit */
```

```
      logData[ 8] = 0;    /* Type Info - logVar2 - Unsigned int, 8 bit */
      logData[ 9] = 0x41; /* Type Info - logVar2 - Unsigned int, 8 bit */
      logData[10] = (uint8)(EventStatusNew); /* EventStatusNew */
#else
      logData[ 0] = 0x42; /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 1] = 0;    /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 2] = 0;    /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 3] = 0;    /* Type Info - logVar1 - Unsigned int, 16 bit */
      logData[ 4] = (uint8)(logVar1);      /* First log variable */
      logData[ 5] = (uint8)(logVar1 >> 8); /* First log variable */
      logData[ 6] = 0x41; /* Type Info - logVar2 - Unsigned int, 8 bit */
      logData[ 7] = 0;    /* Type Info - logVar2 - Unsigned int, 8 bit */
      logData[ 8] = 0;    /* Type Info - logVar2 - Unsigned int, 8 bit */
      logData[ 9] = 0;    /* Type Info - logVar2 - Unsigned int, 8 bit */
      logData[10] = (uint8)(logVar2); /* Second log variable */
#endif

      /* Send log message */
      retVal = Rte_Call_DLTService_4096_SendLogMessage(&logInfo, logData,
                                                       logDataIndex);
      /* Check return value… */
    }
  }
}

/* Do the same for CtDltDemo_ReceiveFrSignal… */
```

4.7.7 DLT master

The DLT requires a master to be triggered and controlled. CANoe and CANape do not have a native support of DLT communication via DltCom. Therefore another tool could be used. For example the DltViewer of GENIVI (http://projects.genivi.org/diagnostic-log-trace/home) can be used.

4.7.8 FIBEX File

The DLT master requires a FIBEX file to interpret the non-verbose messages of DLT correctly. In the payload of a non-verbose message only dynamic data (and the message ID) is set. Thus, in the FIBEX file the corresponding static data, to interpret these dynamic data, is provided.

The FIBEX file is composed of five sections which are explained in Table 4-12. It contains the required content for DEM and DET events and the content of the examples of Table 4-10.

Section	Description

`<?xml version="1.0" encoding="UTF-8" standalone="no"?>`
`<fx:FIBEX xmlns:can="http://www.asam.net/xml/fbx/can"`
`xmlns:fx="http://www.asam.net/xml/fbx"`
`xmlns:ho="http://www.asam.net/xml" VERSION="3.1.0"`
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
`xsi:schemaLocation="http://www.asam.net/xml/fbx`
`xml_schema\fibex.xsd http://www.asam.net/xml/fbx/can`
`xml_schema\fibex4can.xsd">`
`<fx:PROJECT ID="DiagnosticLogAndTrace">`
	Define FIBEX version, project ID and name.

Section	Description
<ho:SHORT-NAME>DLT</ho:SHORT-NAME> </fx:PROJECT> <fx:ELEMENTS>	
<!-- ECU --> <fx:ECUS> <fx:ECU ID="ECUI"> <ho:SHORT-NAME>ECU ID</ho:SHORT-NAME> <fx:MANUFACTURER-EXTENSION> <SW_VERSION>001.000.000</SW_VERSION> <APPLICATIONS> <APPLICATION> <APPLICATION_ID>DEM</APPLICATION_ID> <APPLICATION_DESCRIPTION>DEM event</APPLICATION_DESCRIPTION> <CONTEXTS> <CONTEXT> <CONTEXT_ID>STD0</CONTEXT_ID> <CONTEXT_DESCRIPTION>Dlt_DemTriggerOnEventStatus</CONTEXT_DESCRIPTION> </CONTEXT> </CONTEXTS> </APPLICATION> <APPLICATION> <APPLICATION_ID>DET</APPLICATION_ID> <APPLICATION_DESCRIPTION>DET event</APPLICATION_DESCRIPTION> <CONTEXTS> <CONTEXT> <CONTEXT_ID>STD</CONTEXT_ID> <CONTEXT_DESCRIPTION>Dlt_DetForwardErrorTrace</CONTEXT_DESCRIPTION> </CONTEXT> </CONTEXTS> </APPLICATION> <APPLICATION> <APPLICATION_ID>CtD1</APPLICATION_ID> <APPLICATION_DESCRIPTION>SWC CtDltDemo</APPLICATION_DESCRIPTION> <CONTEXTS> <CONTEXT> <CONTEXT_ID>ReC1</CONTEXT_ID> <CONTEXT_DESCRIPTION>ReceiveCanSignal 1</CONTEXT_DESCRIPTION> </CONTEXT> <CONTEXT> <CONTEXT_ID>ReC2</CONTEXT_ID> <CONTEXT_DESCRIPTION>ReceiveCanSignal 2</CONTEXT_DESCRIPTION> </CONTEXT> </CONTEXTS> </APPLICATION> <APPLICATION> <APPLICATION_ID>CtD2</APPLICATION_ID> <APPLICATION_DESCRIPTION>SWC CtDltDemo</APPLICATION_DESCRIPTION> <CONTEXTS> <CONTEXT> <CONTEXT_ID>ReF1</CONTEXT_ID> <CONTEXT_DESCRIPTION>ReceiveFrSignal 1</CONTEXT_DESCRIPTION> </CONTEXT> <CONTEXT> <CONTEXT_ID>ReF2</CONTEXT_ID> <CONTEXT_DESCRIPTION>ReceiveFrSignal 2</CONTEXT_DESCRIPTION> </CONTEXT> </CONTEXTS> </APPLICATION> </APPLICATIONS> </fx:MANUFACTURER-EXTENSION> </fx:ECU>	Define the ECU ID, the Software version, all application IDs and their corresponding context IDs. In the example on the left, only application ID and the corresponding context ID of DEM events, DET events and the examples of Table 4-10 are defined. More application and context IDs can be added.

Section	Description
</fx:ECUS> <!-- PDUS --> <fx:PDUS> <!--================ Message 1 ==================--> <!--================ 1. Parameter ==================--> <fx:PDU ID="PDU_1_0"> <ho:SHORT-NAME>PDU_1_0</ho:SHORT-NAME> <ho:DESC>Event ID:</ho:DESC> <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> </fx:PDU> <!--================ 2. Parameter ==================--> <fx:PDU ID="PDU_1_1"> <ho:SHORT-NAME>PDU_1_1</ho:SHORT-NAME> <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> <fx:SIGNAL-INSTANCES> <fx:SIGNAL-INSTANCE ID="S_1_0"> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> <fx:SIGNAL-REF ID-REF="S_UINT16"/> </fx:SIGNAL-INSTANCE> </fx:SIGNAL-INSTANCES> </fx:PDU> <!--================ 3. Parameter ==================--> <fx:PDU ID="PDU_1_2"> <ho:SHORT-NAME>PDU_1_2</ho:SHORT-NAME> <ho:DESC>New Event Status:</ho:DESC> <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> </fx:PDU> <!--================ 4. Parameter ==================--> <fx:PDU ID="PDU_1_3"> <ho:SHORT-NAME>PDU_1_3</ho:SHORT-NAME> <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> <fx:SIGNAL-INSTANCES> <fx:SIGNAL-INSTANCE ID="S_1_1"> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> <fx:SIGNAL-REF ID-REF="S_UINT8"/> </fx:SIGNAL-INSTANCE> </fx:SIGNAL-INSTANCES> </fx:PDU> <!--================ Message 2 ==================--> <!--================ 1. Parameter ==================--> <fx:PDU ID="PDU_2_0"> <ho:SHORT-NAME>PDU_2_0</ho:SHORT-NAME> <ho:DESC>Module ID:</ho:DESC> <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> </fx:PDU> <!--================ 2. Parameter ==================--> <fx:PDU ID="PDU_2_1"> <ho:SHORT-NAME>PDU_2_1</ho:SHORT-NAME> <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> <fx:SIGNAL-INSTANCES> <fx:SIGNAL-INSTANCE ID="S_2_0"> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> <fx:SIGNAL-REF ID-REF="S_UINT16"/> </fx:SIGNAL-INSTANCE> </fx:SIGNAL-INSTANCES> </fx:PDU> <!--================ 3. Parameter ==================--> <fx:PDU ID="PDU_2_2"> <ho:SHORT-NAME>PDU_2_2</ho:SHORT-NAME> <ho:DESC>Instance ID:</ho:DESC> <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> </fx:PDU> <!--================ 4. Parameter ==================--> <fx:PDU ID="PDU_2_3"> <ho:SHORT-NAME>PDU_2_3</ho:SHORT-NAME> <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>	Define all PDUs (== Message ID) and their parameter. In the example on the left only DEM events are defined (message ID == 1). The DEM event sends 2 parameter values to DLT master, they are called “S_1_0” and “S_1_0”. To add more information in the message, both parameter values get another parameter with a describing string. Thus, “1. Parameter” provides the string “Event ID:”. “2. Parameter” provides the first received parameter with type uint16. “3. Parameter” provides the string “New Event Status:” and “4. Parameter” provides the corresponding parameter value from type uint8. Messages for DET and the examples of Table 4-10 are also described. More messages and/or parameter can be added if required.

Section	Description

```
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
      <fx:SIGNAL-INSTANCES>
        <fx:SIGNAL-INSTANCE ID="S_2_1">
          <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          <fx:SIGNAL-REF ID-REF="S_UINT8"/>
        </fx:SIGNAL-INSTANCE>
      </fx:SIGNAL-INSTANCES>
    </fx:PDU>
    <!--================ 5. Parameter ==================-->
    <fx:PDU ID="PDU_2_4">
      <ho:SHORT-NAME>PDU_2_4</ho:SHORT-NAME>
      <ho:DESC>API ID:</ho:DESC>
      <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
    </fx:PDU>
    <!--================ 6. Parameter ==================-->
    <fx:PDU ID="PDU_2_5">
      <ho:SHORT-NAME>PDU_2_5</ho:SHORT-NAME>
      <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
      <fx:SIGNAL-INSTANCES>
        <fx:SIGNAL-INSTANCE ID="S_2_2">
          <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          <fx:SIGNAL-REF ID-REF="S_UINT8"/>
        </fx:SIGNAL-INSTANCE>
      </fx:SIGNAL-INSTANCES>
    </fx:PDU>
    <!--================ 7. Parameter ==================-->
    <fx:PDU ID="PDU_2_6">
      <ho:SHORT-NAME>PDU_2_6</ho:SHORT-NAME>
      <ho:DESC>Error ID:</ho:DESC>
      <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
    </fx:PDU>
    <!--================ 8. Parameter ==================-->
    <fx:PDU ID="PDU_2_7">
      <ho:SHORT-NAME>PDU_2_7</ho:SHORT-NAME>
      <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
      <fx:SIGNAL-INSTANCES>
        <fx:SIGNAL-INSTANCE ID="S_2_3">
          <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          <fx:SIGNAL-REF ID-REF="S_UINT8"/>
        </fx:SIGNAL-INSTANCE>
      </fx:SIGNAL-INSTANCES>
    </fx:PDU>
    <!--================ Message 3 ==================-->
    <!--================ 1. Parameter ==================-->
    <fx:PDU ID="PDU_3_0">
      <ho:SHORT-NAME>PDU_3_0</ho:SHORT-NAME>
      <ho:DESC>Log variable 1:</ho:DESC>
      <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
    </fx:PDU>
    <!--================ 2. Parameter ==================-->
    <fx:PDU ID="PDU_3_1">
      <ho:SHORT-NAME>PDU_3_1</ho:SHORT-NAME>
      <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
      <fx:SIGNAL-INSTANCES>
        <fx:SIGNAL-INSTANCE ID="S_3_0">
          <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          <fx:SIGNAL-REF ID-REF="S_UINT16"/>
        </fx:SIGNAL-INSTANCE>
      </fx:SIGNAL-INSTANCES>
    </fx:PDU>
    <!--================ 3. Parameter ==================-->
    <fx:PDU ID="PDU_3_2">
      <ho:SHORT-NAME>PDU_3_2</ho:SHORT-NAME>
      <ho:DESC>Log variable 2:</ho:DESC>
      <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
      <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
    </fx:PDU>
```

Section	Description

```
<!--================ 4. Parameter ==================-->
<fx:PDU ID="PDU_3_3">
  <ho:SHORT-NAME>PDU_3_3</ho:SHORT-NAME>
  <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>
  <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
  <fx:SIGNAL-INSTANCES>
    <fx:SIGNAL-INSTANCE ID="S_3_1">
      <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
      <fx:SIGNAL-REF ID-REF="S_UINT8"/>
    </fx:SIGNAL-INSTANCE>
  </fx:SIGNAL-INSTANCES>
</fx:PDU>
<!--================ Message 4 ==================-->
<!--================ 1. Parameter ==================-->
<fx:PDU ID="PDU_4_0">
  <ho:SHORT-NAME>PDU_4_0</ho:SHORT-NAME>
  <ho:DESC>Log variable 1:</ho:DESC>
  <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
  <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
</fx:PDU>
<!--================ 2. Parameter ==================-->
<fx:PDU ID="PDU_4_1">
  <ho:SHORT-NAME>PDU_4_1</ho:SHORT-NAME>
  <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH>
  <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
  <fx:SIGNAL-INSTANCES>
    <fx:SIGNAL-INSTANCE ID="S_4_0">
      <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
      <fx:SIGNAL-REF ID-REF="S_UINT16"/>
    </fx:SIGNAL-INSTANCE>
  </fx:SIGNAL-INSTANCES>
</fx:PDU>
<!--================ 3. Parameter ==================-->
<fx:PDU ID="PDU_4_2">
  <ho:SHORT-NAME>PDU_4_2</ho:SHORT-NAME>
  <ho:DESC>Log variable 2:</ho:DESC>
  <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
  <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
</fx:PDU>
<!--================ 4. Parameter ==================-->
<fx:PDU ID="PDU_4_3">
  <ho:SHORT-NAME>PDU_4_3</ho:SHORT-NAME>
  <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH>
  <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
  <fx:SIGNAL-INSTANCES>
    <fx:SIGNAL-INSTANCE ID="S_4_1">
      <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
      <fx:SIGNAL-REF ID-REF="S_UINT8"/>
    </fx:SIGNAL-INSTANCE>
  </fx:SIGNAL-INSTANCES>
</fx:PDU>
<!--================ Message 5 ==================-->
<!--================ 1. Parameter ==================-->
<fx:PDU ID="PDU_5_0">
  <ho:SHORT-NAME>PDU_5_0</ho:SHORT-NAME>
  <ho:DESC>Log variable 1:</ho:DESC>
  <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH>
  <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
</fx:PDU>
<!--================ 2. Parameter ==================-->
<fx:PDU ID="PDU_5_1">
  <ho:SHORT-NAME>PDU_5_1</ho:SHORT-NAME>
  <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH>
  <fx:PDU-TYPE>OTHER</fx:PDU-TYPE>
  <fx:SIGNAL-INSTANCES>
    <fx:SIGNAL-INSTANCE ID="S_5_0">
      <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
      <fx:SIGNAL-REF ID-REF="S_UINT16"/>
    </fx:SIGNAL-INSTANCE>
  </fx:SIGNAL-INSTANCES>
</fx:PDU>
<!--================ 3. Parameter ==================-->
<fx:PDU ID="PDU_5_2">
```

Section	Description
<ho:SHORT-NAME>PDU_5_2</ho:SHORT-NAME> <ho:DESC>Log variable 2:</ho:DESC> <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> </fx:PDU> <!--================ 4. Parameter ==================--> <fx:PDU ID="PDU_5_3"> <ho:SHORT-NAME>PDU_5_3</ho:SHORT-NAME> <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> <fx:SIGNAL-INSTANCES> <fx:SIGNAL-INSTANCE ID="S_5_1"> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> <fx:SIGNAL-REF ID-REF="S_UINT8"/> </fx:SIGNAL-INSTANCE> </fx:SIGNAL-INSTANCES> </fx:PDU> </fx:PDUS>	
<!-- FRAMES --> <fx:FRAMES> <!-- 1. Log and Trace Message --> <fx:FRAME ID="ID_1"> <ho:SHORT-NAME>ID_1</ho:SHORT-NAME> <fx:BYTE-LENGTH>3</fx:BYTE-LENGTH> <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE> <fx:PDU-INSTANCES> <fx:PDU-INSTANCE ID="P_1_0"> <fx:PDU-REF ID-REF="PDU_1_0"/> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_1_1"> <fx:PDU-REF ID-REF="PDU_1_1"/> <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_1_2"> <fx:PDU-REF ID-REF="PDU_1_2"/> <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_1_3"> <fx:PDU-REF ID-REF="PDU_1_3"/> <fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> </fx:PDU-INSTANCES> <fx:MANUFACTURER-EXTENSION> <MESSAGE_TYPE>DLT_TYPE_LOG</MESSAGE_TYPE> <MESSAGE_INFO>DLT_LOG_INFO</MESSAGE_INFO> <APPLICATION_ID>DEM</APPLICATION_ID> <CONTEXT_ID>STD0</CONTEXT_ID> <MESSAGE_SOURCE_FILE>Dlt.c</MESSAGE_SOURCE_FILE> <MESSAGE_LINE_NUMBER>1</MESSAGE_LINE_NUMBER> </fx:MANUFACTURER-EXTENSION> </fx:FRAME> <!-- 2. Log and Trace Message --> <fx:FRAME ID="ID_2"> <ho:SHORT-NAME>ID_2</ho:SHORT-NAME> <fx:BYTE-LENGTH>5</fx:BYTE-LENGTH> <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE> <fx:PDU-INSTANCES> <fx:PDU-INSTANCE ID="P_2_0"> <fx:PDU-REF ID-REF="PDU_2_0"/> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_2_1"> <fx:PDU-REF ID-REF="PDU_2_1"/> <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_2_2"> <fx:PDU-REF ID-REF="PDU_2_2"/> <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_2_3"> <fx:PDU-REF ID-REF="PDU_2_3"/> <fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER>	Define all messages with all parameter (also the describing parameters), the message type, the message info, the application ID, the context ID, message source file and the message line number. In the example on the left only the DEM event message is described. All parameter defined in the PDUS-section are listed. The parameter “P_1_1” has type uint16 and parameter “P_1_3” has type uint8, thus the byte length has to be set to 3. The message info can be changed at runtime, thus it is possible that the log level within the ECU differs to the log level in the FIBEX file.

Section	Description

```
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_2_4">
            <fx:PDU-REF ID-REF="PDU_2_4"/>
            <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_2_5">
            <fx:PDU-REF ID-REF="PDU_2_5"/>
            <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_2_6">
            <fx:PDU-REF ID-REF="PDU_2_6"/>
            <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_2_7">
            <fx:PDU-REF ID-REF="PDU_2_7"/>
            <fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
        </fx:PDU-INSTANCES>
        <fx:MANUFACTURER-EXTENSION>
          <MESSAGE_TYPE>DLT_TYPE_LOG</MESSAGE_TYPE>
          <MESSAGE_INFO>DLT_LOG_INFO</MESSAGE_INFO>
          <APPLICATION_ID>DET</APPLICATION_ID>
          <CONTEXT_ID>STD</CONTEXT_ID>
          <MESSAGE_SOURCE_FILE>Dlt.c</MESSAGE_SOURCE_FILE>
          <MESSAGE_LINE_NUMBER>1</MESSAGE_LINE_NUMBER>
        </fx:MANUFACTURER-EXTENSION>
      </fx:FRAME>
      <!-- 3. Log and Trace Message -->
      <fx:FRAME ID="ID_3">
        <ho:SHORT-NAME>ID_3</ho:SHORT-NAME>
        <fx:BYTE-LENGTH>3</fx:BYTE-LENGTH>
        <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE>
        <fx:PDU-INSTANCES>
          <fx:PDU-INSTANCE ID="P_3_0">
            <fx:PDU-REF ID-REF="PDU_3_0"/>
            <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_3_1">
            <fx:PDU-REF ID-REF="PDU_3_1"/>
            <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_3_2">
            <fx:PDU-REF ID-REF="PDU_3_2"/>
            <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_3_3">
            <fx:PDU-REF ID-REF="PDU_3_3"/>
            <fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
        </fx:PDU-INSTANCES>
        <fx:MANUFACTURER-EXTENSION>
          <MESSAGE_TYPE>DLT_TYPE_LOG</MESSAGE_TYPE>
          <MESSAGE_INFO>DLT_LOG_WARN</MESSAGE_INFO>
          <APPLICATION_ID>CtD1</APPLICATION_ID>
          <CONTEXT_ID>ReC1</CONTEXT_ID>

<MESSAGE_SOURCE_FILE>CtDltDemo.c</MESSAGE_SOURCE_FILE>
          <MESSAGE_LINE_NUMBER>1</MESSAGE_LINE_NUMBER>
        </fx:MANUFACTURER-EXTENSION>
      </fx:FRAME>
      <!-- 4. Log and Trace Message -->
      <fx:FRAME ID="ID_4">
        <ho:SHORT-NAME>ID_4</ho:SHORT-NAME>
        <fx:BYTE-LENGTH>3</fx:BYTE-LENGTH>
        <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE>
        <fx:PDU-INSTANCES>
          <fx:PDU-INSTANCE ID="P_4_0">
            <fx:PDU-REF ID-REF="PDU_4_0"/>
            <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_4_1">
            <fx:PDU-REF ID-REF="PDU_4_1"/>
            <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER>
```

Section	Description

```
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_4_2">
            <fx:PDU-REF ID-REF="PDU_4_2"/>
            <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_4_3">
            <fx:PDU-REF ID-REF="PDU_4_3"/>
            <fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
        </fx:PDU-INSTANCES>
        <fx:MANUFACTURER-EXTENSION>
          <MESSAGE_TYPE>DLT_TYPE_LOG</MESSAGE_TYPE>
          <MESSAGE_INFO>DLT_LOG_FATAL</MESSAGE_INFO>
          <APPLICATION_ID>CtD1</APPLICATION_ID>
          <CONTEXT_ID>ReC2</CONTEXT_ID>

<MESSAGE_SOURCE_FILE>CtDltDemo.c</MESSAGE_SOURCE_FILE>
          <MESSAGE_LINE_NUMBER>1</MESSAGE_LINE_NUMBER>
        </fx:MANUFACTURER-EXTENSION>
      </fx:FRAME>
      <!-- 5. Log and Trace Message -->
      <fx:FRAME ID="ID_5">
        <ho:SHORT-NAME>ID_5</ho:SHORT-NAME>
        <fx:BYTE-LENGTH>3</fx:BYTE-LENGTH>
        <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE>
        <fx:PDU-INSTANCES>
          <fx:PDU-INSTANCE ID="P_5_0">
            <fx:PDU-REF ID-REF="PDU_5_0"/>
            <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_5_1">
            <fx:PDU-REF ID-REF="PDU_5_1"/>
            <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_5_2">
            <fx:PDU-REF ID-REF="PDU_5_2"/>
            <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_5_3">
            <fx:PDU-REF ID-REF="PDU_5_3"/>
            <fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
        </fx:PDU-INSTANCES>
        <fx:MANUFACTURER-EXTENSION>
          <MESSAGE_TYPE>DLT_TYPE_LOG</MESSAGE_TYPE>
          <MESSAGE_INFO>DLT_LOG_ERROR</MESSAGE_INFO>
          <APPLICATION_ID>CtD2</APPLICATION_ID>
          <CONTEXT_ID>ReF1</CONTEXT_ID>

<MESSAGE_SOURCE_FILE>CtDltDemo.c</MESSAGE_SOURCE_FILE>
          <MESSAGE_LINE_NUMBER>1</MESSAGE_LINE_NUMBER>
        </fx:MANUFACTURER-EXTENSION>
      </fx:FRAME>
     <!-- 6. Log and Trace Message -->
      <fx:FRAME ID="ID_6">
        <ho:SHORT-NAME>ID_6</ho:SHORT-NAME>
        <fx:BYTE-LENGTH>3</fx:BYTE-LENGTH>
        <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE>
        <fx:PDU-INSTANCES>
          <fx:PDU-INSTANCE ID="P_6_0">
            <fx:PDU-REF ID-REF="PDU_6_0"/>
            <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_6_1">
            <fx:PDU-REF ID-REF="PDU_6_1"/>
            <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_6_2">
            <fx:PDU-REF ID-REF="PDU_6_2"/>
            <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER>
          </fx:PDU-INSTANCE>
          <fx:PDU-INSTANCE ID="P_6_3">
            <fx:PDU-REF ID-REF="PDU_6_3"/>
```

Section	Description
<fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> </fx:PDU-INSTANCES> <fx:MANUFACTURER-EXTENSION> <MESSAGE_TYPE>DLT_TYPE_LOG</MESSAGE_TYPE> <MESSAGE_INFO>DLT_LOG_VERB</MESSAGE_INFO> <APPLICATION_ID>CtD2</APPLICATION_ID> <CONTEXT_ID>ReF2</CONTEXT_ID> <MESSAGE_SOURCE_FILE>CtDltDemo.c</MESSAGE_SOURCE_FILE> <MESSAGE_LINE_NUMBER>1</MESSAGE_LINE_NUMBER> </fx:MANUFACTURER-EXTENSION> </fx:FRAME> </fx:FRAMES>	
<!-- signals --> <fx:SIGNALS> <!-- BOOL --> <fx:SIGNAL ID="S_BOOL"> <ho:SHORT-NAME>S_BOOL</ho:SHORT-NAME> <fx:CODING-REF ID-REF="BOOL"/> </fx:SIGNAL> <!-- SINT8 --> <fx:SIGNAL ID="S_SINT8"> <ho:SHORT-NAME>S_SINT8</ho:SHORT-NAME> <fx:CODING-REF ID-REF="SINT8"/> </fx:SIGNAL> <!-- UINT8 --> <fx:SIGNAL ID="S_UINT8"> <ho:SHORT-NAME>S_UINT8</ho:SHORT-NAME> <fx:CODING-REF ID-REF="UINT8"/> </fx:SIGNAL> <!-- SINT16 --> <fx:SIGNAL ID="S_SINT16"> <ho:SHORT-NAME>S_SINT16</ho:SHORT-NAME> <fx:CODING-REF ID-REF="SINT16"/> </fx:SIGNAL> <!-- UINT16 --> <fx:SIGNAL ID="S_UINT16"> <ho:SHORT-NAME>S_UINT16</ho:SHORT-NAME> <fx:CODING-REF ID-REF="UINT16"/> </fx:SIGNAL> <!-- SINT32 --> <fx:SIGNAL ID="S_SINT32"> <ho:SHORT-NAME>S_SINT32</ho:SHORT-NAME> <fx:CODING-REF ID-REF="SINT32"/> </fx:SIGNAL> <!-- UINT32 --> <fx:SIGNAL ID="S_UINT32"> <ho:SHORT-NAME>S_UINT32</ho:SHORT-NAME> <fx:CODING-REF ID-REF="UINT32"/> </fx:SIGNAL> <!-- SINT64 --> <fx:SIGNAL ID="S_SINT64"> <ho:SHORT-NAME>S_SINT64</ho:SHORT-NAME> <fx:CODING-REF ID-REF="SINT64"/> </fx:SIGNAL> <!-- UINT64 --> <fx:SIGNAL ID="S_UINT64"> <ho:SHORT-NAME>S_UINT64</ho:SHORT-NAME> <fx:CODING-REF ID-REF="UINT64"/> </fx:SIGNAL> <!-- FLOA16 --> <fx:SIGNAL ID="S_FLOA16"> <ho:SHORT-NAME>S_FLOA16</ho:SHORT-NAME> <fx:CODING-REF ID-REF="FLOA16"/> </fx:SIGNAL> <!-- FLOA32 --> <fx:SIGNAL ID="S_FLOA32"> <ho:SHORT-NAME>S_FLOA32</ho:SHORT-NAME> <fx:CODING-REF ID-REF="FLOA32"/> </fx:SIGNAL> <!-- FLOA64 --> <fx:SIGNAL ID="S_FLOA64">	This field is constant and can be copied without adaption. It defines the signal types.

Section	Description
<ho:SHORT-NAME>S_FLOA64</ho:SHORT-NAME> <fx:CODING-REF ID-REF="FLOA64"/> </fx:SIGNAL> <!-- LENGTH --> <fx:SIGNAL ID="S_LENGTH"> <ho:SHORT-NAME>S_LENGTH</ho:SHORT-NAME> <fx:CODING-REF ID-REF="LENGTH"/> </fx:SIGNAL> <!-- STRG_ASCII --> <fx:SIGNAL ID="S_STRG_ASCII"> <ho:SHORT-NAME>S_STRG_ASCII</ho:SHORT-NAME> <fx:CODING-REF ID-REF="STRG_ASCII"/> </fx:SIGNAL> <!-- STRG_UTF8 --> <fx:SIGNAL ID="S_STRG_UTF8"> <ho:SHORT-NAME>S_STRG_UTF8</ho:SHORT-NAME> <fx:CODING-REF ID-REF="STRG_UTF8"/> </fx:SIGNAL> <!-- RAWD --> <fx:SIGNAL ID="S_RAWD"> <ho:SHORT-NAME>S_RAWD</ho:SHORT-NAME> <fx:CODING-REF ID-REF="RAWD"/> </fx:SIGNAL> </fx:SIGNALS> </fx:ELEMENTS>	
<!-- PROCESSING INFORMATION --> <fx:PROCESSING-INFORMATION> <!-- codings --> <fx:CODINGS> <fx:CODING ID="BOOL"> <ho:SHORT-NAME>BOOL</ho:SHORT-NAME> <ho:DESC>Coding for boolean values.</ho:DESC> <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE" ENCODING="UNSIGNED" ho:BASE-DATA-TYPE="A_UINT8"> <ho:BIT-LENGTH>8</ho:BIT-LENGTH> </ho:CODED-TYPE> </fx:CODING> <fx:CODING ID="SINT8"> <ho:SHORT-NAME>SINT8</ho:SHORT-NAME> <ho:DESC>Coding for signed 8bit values.</ho:DESC> <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE" ENCODING="SIGNED" ho:BASE-DATA-TYPE="A_INT8"> <ho:BIT-LENGTH>8</ho:BIT-LENGTH> </ho:CODED-TYPE> </fx:CODING> <fx:CODING ID="UINT8"> <ho:SHORT-NAME>UINT8</ho:SHORT-NAME> <ho:DESC>Coding for unsigned 8bit values.</ho:DESC> <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE" ENCODING="UNSIGNED" ho:BASE-DATA-TYPE="A_UINT8"> <ho:BIT-LENGTH>8</ho:BIT-LENGTH> </ho:CODED-TYPE> </fx:CODING> <fx:CODING ID="SINT16"> <ho:SHORT-NAME>SINT16</ho:SHORT-NAME> <ho:DESC>Coding for signed 16bit values.</ho:DESC> <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE" ENCODING="SIGNED" ho:BASE-DATA-TYPE="A_INT16"> <ho:BIT-LENGTH>16</ho:BIT-LENGTH> </ho:CODED-TYPE> </fx:CODING> <fx:CODING ID="UINT16"> <ho:SHORT-NAME>UINT16</ho:SHORT-NAME> <ho:DESC>Coding for unsigned 16bit values.</ho:DESC> <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE" ENCODING="UNSIGNED" ho:BASE-DATA-TYPE="A_UINT16"> <ho:BIT-LENGTH>16</ho:BIT-LENGTH> </ho:CODED-TYPE> </fx:CODING> <fx:CODING ID="SINT32"> <ho:SHORT-NAME>SINT32</ho:SHORT-NAME> <ho:DESC>Coding for signed 32bit values.</ho:DESC> <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"	This field is constant and can be copied without adaption. It defines the types of the type info field ([Dlt135] of [1]).

Section	Description

```
ENCODING="SIGNED" ho:BASE-DATA-TYPE="A_INT32">
          <ho:BIT-LENGTH>32</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="UINT32">
        <ho:SHORT-NAME>UINT32</ho:SHORT-NAME>
        <ho:DESC>Coding for unsigned 32bit values.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
ENCODING="UNSIGNED" ho:BASE-DATA-TYPE="A_UINT32">
          <ho:BIT-LENGTH>32</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="SINT64">
        <ho:SHORT-NAME>SINT64</ho:SHORT-NAME>
        <ho:DESC>Coding for signed 64bit values.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
ENCODING="SIGNED" ho:BASE-DATA-TYPE="A_INT64">
          <ho:BIT-LENGTH>64</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="UINT64">
        <ho:SHORT-NAME>UINT64</ho:SHORT-NAME>
        <ho:DESC>Coding for unsigned 64bit values.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
ENCODING="UNSIGNED" ho:BASE-DATA-TYPE="A_UINT64">
          <ho:BIT-LENGTH>64</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="FLOA16">
        <ho:SHORT-NAME>FLOA16</ho:SHORT-NAME>
        <ho:DESC>Coding for float 16bit values.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
ENCODING="IEEE-FLOATING-TYPE" ho:BASE-DATA-TYPE="A_FLOAT32">
          <ho:BIT-LENGTH>16</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="FLOA32">
        <ho:SHORT-NAME>FLOA32</ho:SHORT-NAME>
        <ho:DESC>Coding for float 32bit values.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
ENCODING="IEEE-FLOATING-TYPE" ho:BASE-DATA-TYPE="A_FLOAT32">
          <ho:BIT-LENGTH>32</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="FLOA64">
        <ho:SHORT-NAME>FLOA64</ho:SHORT-NAME>
        <ho:DESC>Coding for float 64bit values.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
ENCODING="IEEE-FLOATING-TYPE" ho:BASE-DATA-TYPE="A_FLOAT64">
          <ho:BIT-LENGTH>64</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="LENGTH">
        <ho:SHORT-NAME>LENGTH</ho:SHORT-NAME>
        <ho:DESC>Coding for length information.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
ENCODING="UNSIGNED" ho:BASE-DATA-TYPE="A_UINT16">
          <ho:BIT-LENGTH>16</ho:BIT-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="STRG_ASCII">
        <ho:SHORT-NAME>STRG_ASCII</ho:SHORT-NAME>
        <ho:DESC>Coding for ASCII string.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
TERMINATION="ZERO" ho:BASE-DATA-TYPE="A_ASCIISTRING">
          <ho:MIN-LENGTH>0</ho:MIN-LENGTH>
          <ho:MAX-LENGTH>5120</ho:MAX-LENGTH>
        </ho:CODED-TYPE>
      </fx:CODING>
      <fx:CODING ID="STRG_UTF8">
        <ho:SHORT-NAME>STRG_UTF8</ho:SHORT-NAME>
        <ho:DESC>Coding for UTF8 string.</ho:DESC>
        <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE"
```

Section	Description
ENCODING="UTF-8" TERMINATION="ZERO" ho:BASE-DATA-TYPE="A_UNICODE2STRING"> <ho:MIN-LENGTH>0</ho:MIN-LENGTH> <ho:MAX-LENGTH>5120</ho:MAX-LENGTH> </ho:CODED-TYPE> </fx:CODING> <fx:CODING ID="RAWD"> <ho:SHORT-NAME>RAWD</ho:SHORT-NAME> <ho:DESC>Coding for raw data.</ho:DESC> <ho:CODED-TYPE CATEGORY="STANDARD-LENGTH-TYPE" ENCODING="UNSIGNED" ho:BASE-DATA-TYPE="A_BYTEFIELD"> <ho:MIN-LENGTH>0</ho:MIN-LENGTH> <ho:MAX-LENGTH>5120</ho:MAX-LENGTH> </ho:CODED-TYPE> </fx:CODING> </fx:CODINGS> </fx:PROCESSING-INFORMATION> </fx:FIBEX>	

Table 4-12 Structure of FIBEX file (complete FIBEX file with one example)

4.7.9 VFB tracing

VFB tracing is the implicit tracing of communication between SWCs via RTE. The SWCs do not send trace messages explicitly. Therefore, the RTE supports hook functions which are called before and after the execution of a RTE service. These hooks can be individually activated and deactivated within the DaVinci Configurator Pro by `/MICROSAR/Rte/RteGeneration/RteVfbTraceFunction`. To use them the global switch `/MICROSAR/Rte/RteGeneration/RteVfbTraceEnabled` has to be set.

It is not required to configure the DaVinci Developer.

Note
It should be sufficient to use the start **or** the return hook for VFB tracing.

Caution
VFB tracing is not supported explicitly, but it is possible to implement it manually. Thus, it is not required to activate the DaVinci Configurator Pro switch `/MICROSAR/Dlt/DltGeneral/DltImplementVfbTrace`.

The chosen hook functions have to be implemented manually, only the declaration is available in Rte_Hook.h.

4.7.9.1 Context Registration for VFB tracing

There are two different ways to register the contexts of VFB tracing.

4.7.9.1.1 Registration from hook

Precondition for this method is the following configuration:

> In DaVinci Configurator Pro for each VFB trace context one `/MICROSAR/Dlt/DltNonVerboseMessage` (= message ID) has to be added. No more configuration has to be done (do not increment the number of max DLT users).

> In DaVinci Developer no configuration has to be done.

Within the hook function the DLT user context has to be registered once. Therefore call Dlt_RegisterContext directly. The first expected parameter is the session ID. For SWC this parameter is set by RTE, in case of VFB tracing the session ID has to be set manually. Choose the session ID higher than 4096 and unequal to the already used session IDs.

Table 4-13 shows all parameter to be passed to Dlt_RegisterContext.

Parameter	Recommended Value	Description
Parg0 (session ID)	> 4096	Value must be greater than 4096 and unique in the ECU. One session ID for all VFB trace contexts is sufficient.
AppId	“VFBT” = 0x56464254	The application ID for VFB tracing has to be the four ASCII letters “VFBT”.
ContextId	wxyz	The context ID has to be chosen uniquely for the VFB tracing.
AppDescription	NULL_PTR	The application ID description is not supported.
LenAppDescription	0	Must be 0 if AppDescription is NULL_PTR.
ContextDescription	NULL_PTR	The context ID description is not supported.
LenContextDescription	0	Must be 0 if ContextDescription is NULL_PTR.

Table 4-13 Recommended Parameter of Dlt_RegisterContext for VFB tracing

With this method the VFB tracing is clearly distinct from the logging and tracing of SWCs and it is easier to configure. But each time a hook using VFB tracing is called, the context has to be checked if it is already registered. This increases the runtime of RTE.

4.7.9.1.2 Registration from SWC

To avoid the check if the context is already registered within the hook function, it is possible to register the VFB trace context within an init runnable of a SWC. Therefore a separated DLT user (= Session ID) should be created, thus the used VFB trace hooks have their own session ID.

Required configuration in DaVinci Configurator Pro:

> add one /MICROSAR/Dlt/DltNonVerboseMessage (= message ID) per VFB trace context

> add one /MICROSAR/Dlt/DltGeneral/DltMaxNumberOfDltUsers (= session ID; one is sufficient)

Required configuration in DaVinci Developer:

> Add an init runnable to the chosen SWC (if it does not already exist)

> Add a service port to the SWC and name it LogTraceSessionControl_<SessionID> (must have Server direction, the name may differ)

> For this session ID no service port with client direction has to be added (Dlt_SendTraceMessage is not send from SWC)

> Add three runnables to the SWC which are triggered by (LogTraceSessionControl_<SessionID>.SetLogLevel, .SetTraceStatus, .SetVerboseMode)

The parameters that have to be used for VFB trace context registration are described in Table 4-13.

This method increases configuration effort, but reduces implementation effort and runtime of RTE.

Note
The session ID, application ID and context ID of context registration has to be used for sending trace messages in the hook function.

4.7.9.2 Sending VFB trace messages

After the succeeded context registration, the function Dlt_SendTraceMessage can be called to send a VFB tracing message. This function has to be called within the hook function.

Usually all parameter of the hook function have to be concatenated in the trace data, but it is also possible to disclaim some or all parameter. At least the message ID (first parameter in trace data) has to be set.

In Table 4-14 shows all parameter to be passed to Dlt_SendTraceMessage.

Parameter	Recommended Value	Description
Parg0 (session ID)	> 4096	The registered session ID for this VFB trace context.
TraceInfo .app_id .conId .options .trace_info	=“VFBT” =wxyz =DLT_NON_VERBOSE_MSG | DLT_TYPE_NW_TRACE =DLT_NW_TRACE_IPC	Use always “VFBT” as application ID. Use the hook specific context ID. Set option to “non-verbose” and the message type to “network trace”. Set the trace info to “inter process communication”.
TraceData	=<MessageId><HookParameter0><HookParameter1><…>	Concatenate firstly the message ID (must be unique in ECU) and then all required parameter. The order of parameter has to be considered in the FIBEX file.
TraceDataLength	4-65535	At least the message ID has to be passed in the trace data.

Table 4-14 Recommended Parameter of Dlt_SendTraceMessage for VFB tracing

`TraceData` enables to send individual data. Normally it is used to send the parameter values of the hook function's signature. The number of parameter can vary in number and type of each hook function. There is no specification which parameter comes first (except the message ID, which is always first). Thus the order of parameter can be freely chosen, but it has to be according the specified messages in the FIBEX file.

In Table 4-15 one VFB trace message is defined in the FIBEX file. Due to this data, the hook function has two parameters. The first is of type uint16 and the second is of type uint8. To interpret the receiving data correctly, the payload of VFB trace message must contain 4Byte message ID, 2Byte data of parameter one and 1Byte data of parameter two.

Section	Description
<!-- ECU --> <fx:ECUS> <fx:ECU ID="ECUI"> <ho:SHORT-NAME>ECU ID</ho:SHORT-NAME> <fx:MANUFACTURER-EXTENSION> <SW_VERSION>001.000.000</SW_VERSION> <APPLICATIONS> <APPLICATION> <APPLICATION_ID>VFBT</APPLICATION_ID> <APPLICATION_DESCRIPTION>VFB tracing</APPLICATION_DESCRIPTION> <CONTEXTS> <CONTEXT> <CONTEXT_ID>EXAM</CONTEXT_ID> <CONTEXT_DESCRIPTION>RteCallHook </CONTEXT_DESCRIPTION> </CONTEXT> </CONTEXTS> </APPLICATION> </APPLICATIONS> </fx:MANUFACTURER-EXTENSION> </fx:ECU> </fx:ECUS>	In the example on the left, only application ID and the corresponding context ID of one VFB trace event is defined. The application ID (=VFBT) is fixed. The context ID depends on the context ID in the ECU.
<!-- PDUS --> <fx:PDUS> <!-- Message 3 --> <!-- 1. Parameter --> <fx:PDU ID="PDU_3_0"> <ho:SHORT-NAME>PDU_3_0</ho:SHORT-NAME> <ho:DESC>Parameter_1_Describtion:</ho:DESC> <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> </fx:PDU> <!-- 2. Parameter --> <fx:PDU ID="PDU_3_1"> <ho:SHORT-NAME>PDU_3_1</ho:SHORT-NAME> <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> <fx:SIGNAL-INSTANCES> <fx:SIGNAL-INSTANCE ID="S_3_0"> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> <fx:SIGNAL-REF ID-REF="S_UINT16"/> </fx:SIGNAL-INSTANCE> </fx:SIGNAL-INSTANCES> </fx:PDU> <!-- 3. Parameter --> <fx:PDU ID="PDU_3_2"> <ho:SHORT-NAME>PDU_3_2</ho:SHORT-NAME> <ho:DESC>Parameter_2_Describtion:</ho:DESC> <fx:BYTE-LENGTH>0</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> </fx:PDU> <!-- 4. Parameter --> <fx:PDU ID="PDU_3_3">	Define all PDUs (== Message ID) and their parameter. In the example on the left only DEM events are defined (message ID == 1). The DEM event sends 2 parameter values to DLT master, they are called "S_1_0" and "S_1_0". To add more information in the message, both parameter values get another parameter with a describing string. Thus, "1. Parameter" provides the string "Event ID:". "2. Parameter" provides the first received parameter with type uint16. "3. Parameter" provides the string "New Event Status:" and "4. Parameter" provides the corresponding parameter value

Section	Description
<ho:SHORT-NAME>PDU_3_3</ho:SHORT-NAME> <fx:BYTE-LENGTH>1</fx:BYTE-LENGTH> <fx:PDU-TYPE>OTHER</fx:PDU-TYPE> <fx:SIGNAL-INSTANCES> <fx:SIGNAL-INSTANCE ID="S_3_1"> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> <fx:SIGNAL-REF ID-REF="S_UINT8"/> </fx:SIGNAL-INSTANCE> </fx:SIGNAL-INSTANCES> </fx:PDU> </fx:PDUS>	from type uint8. More messages and/or parameter can be added if required.
<!-- FRAMES --> <fx:FRAMES> <!-- 1. Log and Trace Message --> <fx:FRAME ID="ID_3"> <ho:SHORT-NAME>ID_3</ho:SHORT-NAME> <fx:BYTE-LENGTH>3</fx:BYTE-LENGTH> <fx:FRAME-TYPE>OTHER</fx:FRAME-TYPE> <fx:PDU-INSTANCES> <fx:PDU-INSTANCE ID="P_3_0"> <fx:PDU-REF ID-REF="PDU_3_0"/> <fx:SEQUENCE-NUMBER>0</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_3_1"> <fx:PDU-REF ID-REF="PDU_3_1"/> <fx:SEQUENCE-NUMBER>1</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_3_2"> <fx:PDU-REF ID-REF="PDU_3_2"/> <fx:SEQUENCE-NUMBER>2</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> <fx:PDU-INSTANCE ID="P_3_3"> <fx:PDU-REF ID-REF="PDU_3_3"/> <fx:SEQUENCE-NUMBER>3</fx:SEQUENCE-NUMBER> </fx:PDU-INSTANCE> </fx:PDU-INSTANCES> <fx:MANUFACTURER-EXTENSION> <MESSAGE_TYPE>DLT_TYPE_NW_TRACE</MESSAGE_TYPE> <MESSAGE_INFO>DLT_NW_TRACE_IPC</MESSAGE_INFO> <APPLICATION_ID>VFBT</APPLICATION_ID> <CONTEXT_ID>EXAM</CONTEXT_ID> <MESSAGE_SOURCE_FILE>Dlt.c</MESSAGE_SOURCE_FILE> <MESSAGE_LINE_NUMBER>1</MESSAGE_LINE_NUMBER> </fx:MANUFACTURER-EXTENSION> </fx:FRAME> </fx:FRAMES>	Define frame with message ID == 3 (no other messages are used). The message type has to be DLT_TYPE_NW_TRACE and the message info has to be DLT_NW_TRACE_IPC.

Table 4-15 FIBEX file content for one VFB trace message

Caution
The parameter order in the payload of a trace/log messages have to be equal to the parameter order in the FIBEX file.

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 controls the state machine if DltCom is used. Otherwise it 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_SendTraceMessage

Prototype
`Dlt_ReturnType` **`Dlt_SendTraceMessage`** `( Dlt_SessionIDType SessionId, P2CONST(Dlt_MessageTraceInfoType, AUTOMATIC, DLT_APPL_VAR) TraceInfo, P2CONST(uint8, AUTOMATIC, DLT_APPL_VAR) TraceData, uint16 TraceDataLength)`

Parameter	
SessionId	For SW-C this is not visible (Port defined argument value), for BSW-modules it is the module number. This parameter is specified by AUTOSAR but is current not used by the implementation.
TraceInfo	Structure containing information whether the message shall be transmitted in verbose or non-verbose mode and filtering information.
TraceData	Buffer containing the parameters to be logged. The content of this pointer represents the payload of the send log message.
TraceDataLength	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 TraceInfo->trace_info is in valid range. The range depends on TraceInfo->options (DLT_TYPE_APP_TRACE || DLT_TYPE_NW_TRACE). 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. > Tracing 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 5-9 Dlt_SendTraceMessage

5.2.9 Dlt_RegisterContext

Prototype

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

Parameter

SessionId	Number of the module (Module ID within BSW, Port defined argument value within SW-C)
app_id	The Application ID.
context_id	The Context ID
app_description	Points to description string for the provided application id. At maximum 255 characters are interpreted.
app_description_len	The length of the description for the application id string (number of characters of description string).
context_description	Points to description string for the provided context. At maximum 255 characters are interpreted.
context_description_len	The length of the description string (number of characters of description string).

Return code

Dlt_ReturnType	DLT_E_CONTEXT_ALREADY_REG - The software module context has already been registered. DLT_E_ERROR_TO_MANY_CONTEXT – There were too many registration of contexts for one application ID, too many registrations of application IDs or too many registrations of overall context IDs. DLT_E_UNKNOWN_SESSION_ID - The session ID is not valid. DLT_E_OK - The required operation succeed.

Functional Description

Registers the DLT users' context.

The service has to be called when a software module wants to use services offered by DLT software component for a specific context. If a context id is being registered for an already registered application id then app_description can be NULL and len_app_description zero.

Particularities and Limitations

- This function is not reentrant.
- This function is synchronous.

Call context
> This function can be called in any context.

Table 5-10 Dlt_RegisterContext

5.2.10 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 5-11 Dlt_SetState

5.2.11 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 5-12 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 SchM_Exit_Dlt
APPL	Appl_DltDemEventFilterCbk

Table 5-13 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