VECTOR

XCP on FlexRay

Technical Reference

Version 1.13.00

Status	Released

1 Document Information

1.1 History

Date	Version	Remarks
2007-06-11	1.00.00	Creation of document
2007-07-09	1.01.00	Support of AUTOSAR Memory Mapping
2007-07-30	1.02.00	New GENy GUI features
2007-09-10	1.03.00	Update for Vector-Release
2008-02-25	1.04.00	New Feature “Use Tx Confirmation”
2009-02-09	1.05.00	Support of Ecuc Import/Export
2009-09-09	1.06.00	Support a2l export
2010-08-25	1.07.00	ESCAN00044862: Multiple Transport Layer support
2010-12-10	1.08.00	ESCAN00046308: AR3-297 AR3-894: Support PduInfoType instead of the DataPtr
2011-08-11	1.09.00	Support of Post Build configuration variant
2012-11-09	1.10.00	Added Option for AMD Runtime Measurement ESCAN00058756 Describe use case for PDU length ESCAN00065888 Cluster Name should be Cluster ID ESCAN00066648 Describe usage of new Exclusive Areas
2013-09-10	1.11.00	ESCAN00070207 Optimization - 32bit copy ESCAN00070081 Describe resume mode ESCAN00069779 Missing description of "Set PduMode Support" Feature in GENy ESCAN00067440 Max CTO must not be greater than Max DTO
2014-08-15	1.11.01	ESCAN00077234 AR3-2679: Description BCD-coded return-value of FrXcp_GetVersionInfo() in TechRef
2015-05-05	1.12.00	Removed chapter GENy ESCAN00083373 Tx Confirmation Timeout Timer ESCAN00083375 Missing entries in root config structure
2016-10-13	1.13.00	ESCAN00092303 FrXcp_Control API replaced by global Variable
2019-04-04	2.00.00	Added protocol layer API for MainFunction forwarding

Table 1-1 History of the Document

1.2 Reference Documents

Index and Document Name
[1] XCP -Part 2- Protocol Layer Specification -1.1.pdf
[2] XCP -Part 3- Transport Layer Specification XCP on FlexRay -1.1.pdf
[3] API specification of Development Error Tracer, Version 1.0.0 of 2005-07-08
[4] Specification of Platform Types Version 1.1.0 of 2005-04-29

[5]	Specification of Standard Types Version 1.0.3 of 2005-12-13
[6]	TechnicalReference_Asr_Xcp.pdf

Table 1-2 Reference Documents

1.3 Scope of this document

This document describes the features, API, configuration and integration of the XCP Transport Layer for FlexRay. The XCP Protocol Layer, which is already described within a separate document [6], is not covered by this document.

Please also refer to “The Universal Measurement and Calibration Protocol Family” specification by ASAM e.V.

Contents

Illustrations

Tables

2 Overview

XCP on FlexRay is a hardware independent, yet bus specific protocol that can be ported to almost any FlexRay controller. Due to there are numerous combinations of micro controllers, compilers, FlexRay stacks and memory models it cannot be guaranteed that it will run properly on all of the above mentioned combinations.

Please note that in this document the term Application is not used strictly for the user software but also for any higher software layer, like e.g. a Communication Control Layer. Therefore, Application refers to any of the software components using XCP on FlexRay.

The API of the functions is described in a separate chapter at the end of this document.

2.1 Abbreviations and Items

Abbreviations	Complete expression
A2L	File Extension for an **A**SAM **2**MC **L**anguage File
AML	**A**SAM 2 **M**eta **L**anguage
API	**A**pplication **P**rogramming **I**nterface
ASAM	**A**ssociation for **S**tandardization of **A**utomation and **M**easuring Systems
CAN	**C**ontroller **A**rea **N**etwork
CANape	Calibration and Measurement Data Acquisition for Electronic Control Systems
CC	**C**ommunication **C**ontroller
CMD	**C**o**m**man**d**
CTO	**C**ommand **T**ransfer **O**bject
DAQ	Synchronous **D**ata **A**c**q**uistion
DLC	**D**ata **L**ength **C**ode (Number of data bytes of a CAN message)
DLL	**D**ata **l**ink **l**ayer
DTO	**D**ata **T**ransfer **O**bject
ECU	**E**lectronic **C**ontrol **U**nit
ID	**Id**entifier (of a CAN message)
Identifier	Identifies a CAN message
ISR	**I**nterrupt **S**ervice **R**outine
MCS	**M**aster **C**alibration **S**ystem
Message	One or more signals are assigned to each message.
MRB	**M**ulti **r**eceive **b**uffer
MRC	**M**ulti **r**eceive **c**hannel
OEM	**O**riginal **e**quipment **m**anufacturer (vehicle manufacturer)
RES	Command **Res**ponse Packet
SRB	**S**ingle **r**eceive **b**uffer
SERV	**Ser**vice Request Packet
STIM	**Stim**ulation

XCP	Universal Measurement and **C**alibration **P**rotocol
VI	**V**ector **I**nformatik GmbH

Table 2-1 Abbreviations and Items

2.2 Naming Conventions

Naming conventions	
_Asr	AUTOSAR conformant Implementation
_Vector	Vector specific implementation
Fr_	Services of FlexRay Driver
FrIf_	Services of FlexRay Interface
FrNm_	Services of FlexRay Network Management
FrNodeM_	Services of FlexRay Node Manager
FrTP_	Services of FlexRay Transport Layer
FrTrcv_	Services of FlexRay Transceiver Driver
Nm_	Services of Generic NM.
ApplXcp_	Services of XCP Hardware Abstraction Layer on FlexRay
Xcp_	Services of XCP Protocol Layer on FlexRay
FrXcp_	Services of XCP Transport Layer on FlexRay

Table 2-2 Naming Conventions

The names of the service functions start with a prefix that denominates the software component where the service is located. E.g. a service that starts with '`XcpPl_`' is implemented within the XCP Protocol Layer.

In case of callback functions a term that denominates the component that calls the callback function is appended. E.g. '`ComM_Nm_`' denominates a service, which is implemented within the COM Manager and called by Generic NM.

2.3 Architecture Overview

2.3.1 XCP Architecture

Up to now XCP consists of three modules:

- Hardware Abstraction Layer[1]
- XCP Protocol Layer[1]
- XCP Transport Layer

The XCP Protocol Layer provides generic XCP functionality independent from the underlying bus interface while the XCP Transport Layer provides the bus dependent part, e.g. XCP on FlexRay. It can easily be substituted by a XCP on CAN or XCP on LIN Transport Layer. Interaction with the application is done exclusively by the XCP Protocol Layer.

[1] Not covered by this document.

The following figure shows the software architecture of XCP that comprises the XCP Protocol Layer (XcpProf), the XCP Transport Layer (FrXcp), and a hardware abstraction Layer (xcp_appl).

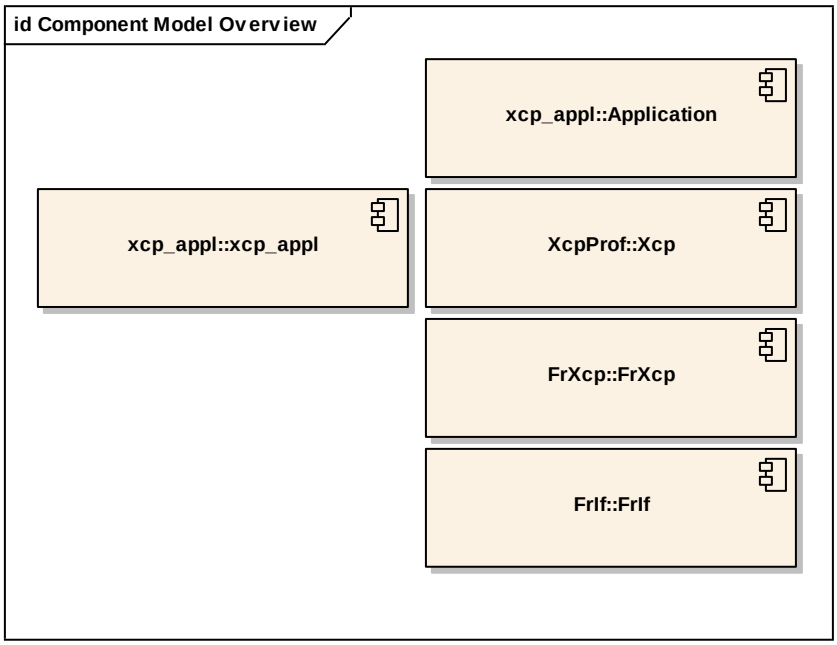

Figure 2-1 XCP Architecture

2.3.2 Detailed Architecture of XCP

The XCP Transport Layer uses the FlexRay Interface as underlying layer for transmission and reception of XCP frames.

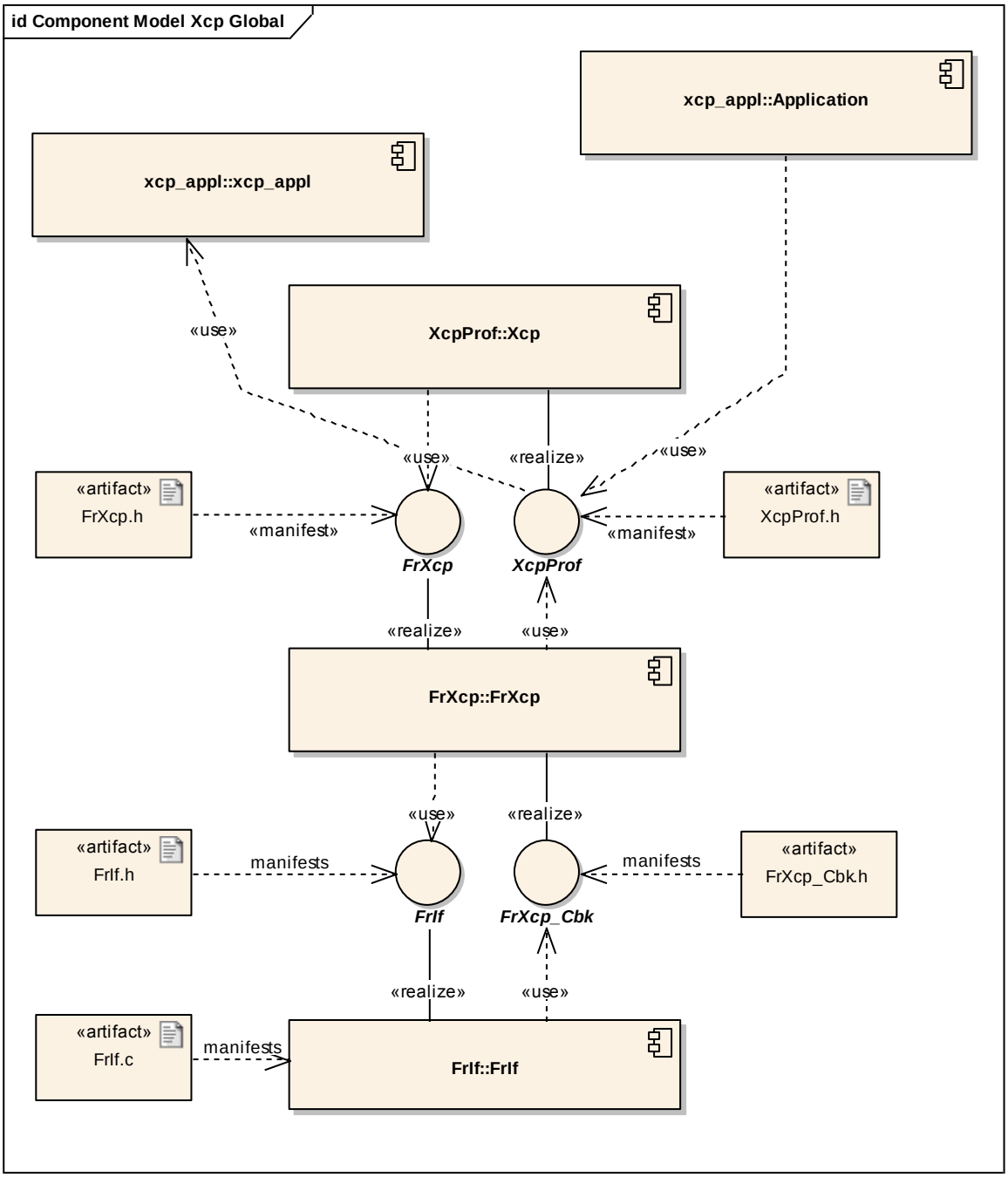

Figure 2-2 Detailed Architecture of XCP

2.3.3 Include structure

The XCP Transport Layer uses the Standard AUTOSAR include structure. Therefore the header Std_Types.h is required. See 4.2 for further details.

3 Functional Description

3.1 Overview of the Functional Scope

The XCP Transport Layer manages transmission and reception of XCP frames. It is also responsible for abstraction of possible bus dependencies. For this purpose and to make maximum use of FlexRay the Transport Layer supports features like frame concatenation, alignment of XCP frames to architectural needs and reconfiguration of used PDUs as described in [2].

The following API is provided by the XCP Transport Layer:

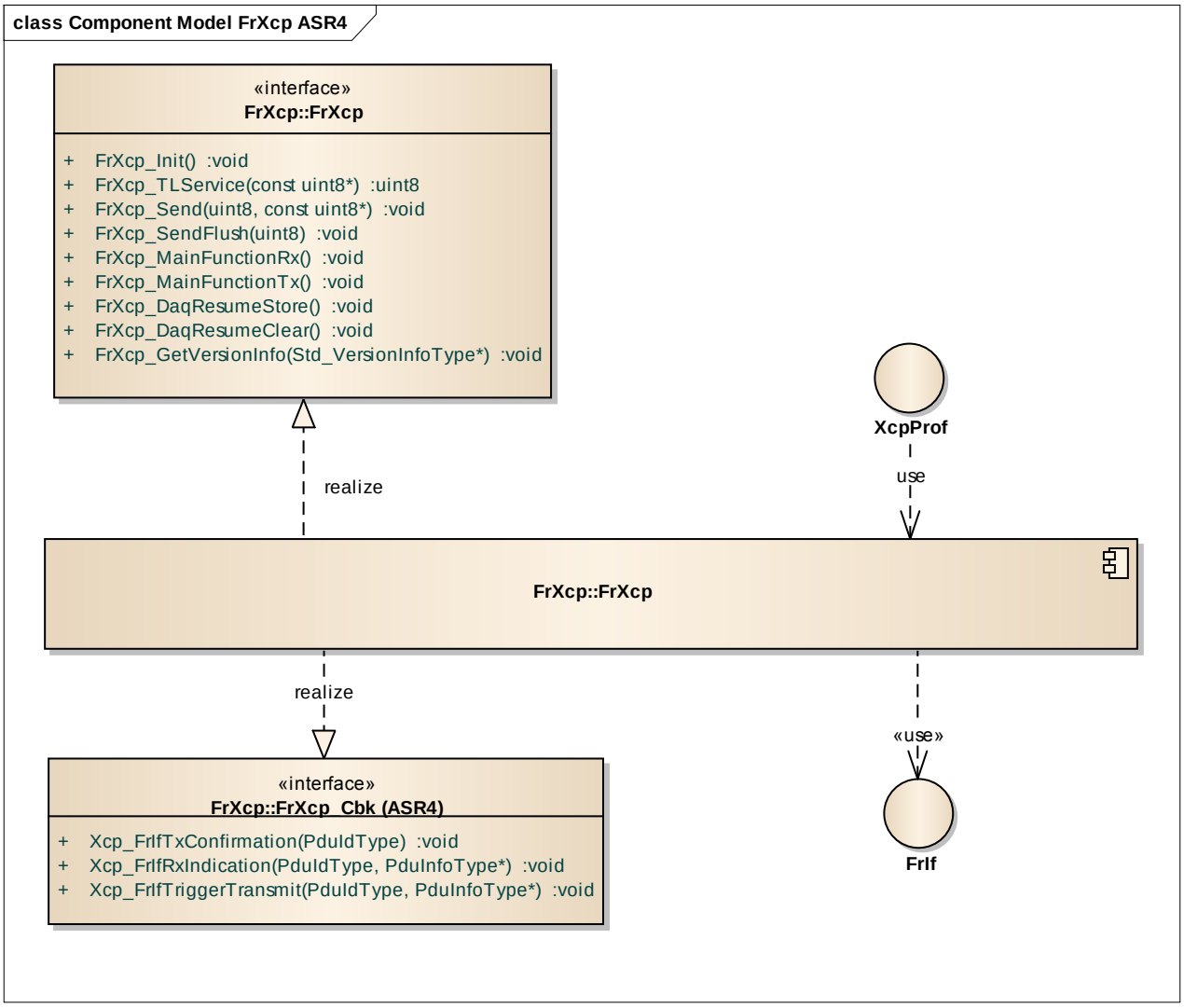

Figure 3-1 API of XCP on FlexRay Transport Layer

Out of these public visible functions, only the FrXcp_MainFunctionRx and FrXcp_MainFunctionTx are relevant for the application. These services must be called cyclically for XCP message handling, see (4.5 Main Functions).
All other services are called **exclusively** by the XCP Protocol Layer.

4 Integration into the Application

This chapter describes the steps necessary for successful integration of the XCP Transport Layer for FlexRay into an application environment of an ECU.

4.1 XCP Transport Layer Files

The files required for the XCP Transport Layer depend on whether source code or object code is available:

Files of XCP on FlexRay Transport Layer with object code delivery		

`FrXcp.o`
`FrXcp.obj`
`FrXcp.lib`
	/ XCP Transport Layer object code / library.
	/ This file must not be modified by the user!
		

`FrXcp.h`	API definitions of the XCP Transport Layer FlexRay. This file must not be modified by the user!	
`FrXcp_Cbk.h`	API of XCP Transport Layer call-back functions. This file must not be modified by the user!	
`FrXcp_Cfg.h`	Pre-compile time configuration header file for the XCP Transport Layer.	
`FrXcp_Lcfg.c`	Link time configuration source file for the XCP Transport Layer.	
`FrXcp_Lcfg.h`	Link time configuration header file for the XCP Transport Layer.	

Table 4-1 File List of XCP on FlexRay (object code)

Files of XCP on FlexRay Transport Layer with source code delivery		
`FrXcp.c`	XCP Transport Layer source code. This file must not be modified by the user!	
`FrXcp.h`	XCP Transport Layer API definitions. This file must not be modified by the user!	
`FrXcp_Cbk.h`	API of XCP Transport Layer call-back functions. This file must not be modified by the user!	
`FrXcp_Cfg.h`	Pre-compile time configuration header file for the XCP Transport Layer.	
`FrXcp_Lcfg.c`	Link time configuration source file for the XCP Transport Layer.	

FrXcp_Lcfg.h	Link time configuration header file for the XCP Transport Layer.	

Table 4-2 File List of XCP on FlexRay (source code)

4.2 Auxiliary Files

The following files are not part of the XCP Transport Layer. They are, however, used by the Transport Layer and must be provided accordingly.

Auxiliary Files used by the XCP on FlexRay		
Std_Types.h	Standard AUTOSAR header.	
ComStack_Types.h	Standard AUTOSAR header.	
FrIf.h	FlexRay Interface header, providing API prototypes for XCP on FlexRay.	
Det.h	Development Error Tracer header, providing API prototypes for XCP on FlexRay.	
MemMap.h	AUTOSAR Memory Mapping header (optional). This header is used if memory mapping is enabled. It must be provided by the user to define the section mapping.	

Table 4-3 Auxiliary File List for XCP on FlexRay

4.3 Version Changes

Changes and the release versions of the XCP on FlexRay Transport Layer are listed at the beginning of the header and source code.

4.4 Initialization

For initialization the service FrXcp_Init must be called. In Pre-compile and Link time configurations a NULL_PTR must be passed. In Post build configuration or multi config the pointer of the Root config struct must be passed.

4.5 Main Functions

The XCP Protocol Layer as well as the XCP Transport Layer have main functions that must be called cyclically.

Please refer to [6] for a detailed description of the XCP Protocol Layer main function

- FrXcp_MainFunctionRx
- FrXcp_MainFunctionTx

The XCP Transport Layer provides two main functions. One is responsible for handling reception related processing and one is responsible for transmission processing. If the respective configuration option “Use Tx Task” or “Use Rx Task” is activated in the Generation Tool the MainFunction must be used accordingly. The `FrXcp_MainFunctionRx` must be executed after the FrIf Job that processes the FrXcp_RxIndication, the `FrXcp_MainFunctionTx` before the FrIf Job that processes the FrXcp_TriggerTransmit.

4.6 Critical Sections

The XCP makes use of interrupt locking to guarantee atomic operation of critical sections. For this purpose three exclusive areas are defined

- FRXCP_EXCLUSIVE_AREA_0
- FRXCP_EXCLUSIVE_AREA_1
- FRXCP_EXCLUSIVE_AREA_2

These three exclusive areas must be mapped to interrupt lock and unlock functions which can be called nested. The exclusive areas are used in the following cases:

4.6.1 FRXCP_EXCLUSIVE_AREA_0

This area is used whenever the services `Xcp_Event`, `Xcp_SendCallBack`, `Xcp_MainFunction` and `Xcp_Command` can interrupt each other.

4.6.2 FRXCP_EXCLUSIVE_AREA_1

This area is used during the `FrXcp_MainFunctionRx`. When it is guaranteed that this MainFunction cannot be interrupted by the `FrXcp_RxIndication` this exclusive area can be left empty, i.e. not mapped to an interrupt disable function.

4.6.3 FRXCP_EXCLUSIVE_AREA_2

This area is used during the `FrXcp_MainFunctionTx`. When it is guaranteed that this MainFunction cannot be interrupted by the `FrXcp_TxConfirmation` or `FrXcp_TriggerTransmit` this exclusive area can be left empty, i.e. not mapped to an interrupt disable function.

4.7 PDU Mode

The FrXcp has an API to disable PDU Transmission called: `FrXcp_SetPduMode()`. This API can be used to prevent transmission of PDUs when set to `FRXCP_SET_OFFLINE`. The PDUs are not lost but stored in the Send Queue until an overrun occurs while in the offline phase. If the PDU Mode is set to online again transmission of PDUs is resumed.
This feature is useful when measurement during bus offline phases shall be performed, for example in the Resume Mode. Measurement is started automatically after Init while the FlexRay Bus still needs a few ms to go sync. During this time, measurement data is stored and transmission is started when the bus is sync.
In a MICROSAR 3 Stack this API is automatically used by the FrSm, therefore no calls by the application are required. When a non MICROSAR FrSm is used or in an AUTOSAR 4 environment the API must be called manually, if activated.

4.8 PDUs

The PDU ID for the underlying Layer can be freely configured in the GenTool respectively in the configuration files. The XCP Transport Layer itself expects an incrementing PDU-ID for each PDU, starting with 0 for the first PDU. This holds true for `FrXcp_RxIndication` and `FrXcp_TriggerTransmit` respectively `FrXcp_TxConfirmation`.

The PDUs need a size of at least 8 bytes if the buffer types are static. In most cases they are dynamic, i.e. reconfigured during runtime. In this case the PDUs need a size of at least 12 bytes.

4.9 Initialized Memory

In order for the DET to work correctly, initialized memory is required. That means that all RAM, used by the XCP Transport Layer, has to be zeroed out. This is usually done by the startup code. If the startup code does not do this the service `FrXcp_InitMemory` must be called

4.10 Memory Mapping

The XCP supports the AUTOSAR Memory Mapping. The following standard sections are used by the Transport Layer as well as the Protocol Layer:

- `SEC_CODE`
- `SEC_CONST_8BIT`
- `SEC_CONST_16BIT`
- `SEC_CONST_UNSPECIFIED`
- `SEC_VAR_NOINIT_UNSPECIFIED`
- `SEC_VAR_NOINIT_8BIT`

4.11 Activation Macros

The XCP Protocol Layer and Transport Layer can be activated and deactivated by a single macro called `XCP_ACTIVATE()` and `XCP_DEACTIVATE()`. By default XCP is activated.

This feature can be used to en- or disable the XCP module during run time. Thus XCP functionality can be controlled by the application, e.g. by a diagnostic service to enable the component. This feature allows the component to remain in series production ECU's where it can be enabled on demand.

4.12 Integration Notes

4.12.1 Alignment

Depending on the used Hardware it might be required to use an Alignment setting > 8Bit. The following Table gives an example overview of several Architectures and their alignment requirements:

Architecture	8Bit Alignment	16Bit Alignment	32Bit Alignment

MCS12X	Yes	Yes[1]	Yes[1]
TriCore	No	Yes[2]	Yes
V850 FX2	No	No	Yes
V850 Phoenix FS	Yes[2]	Yes[2]	Yes

4.12.2 CANape Buffer ID

The Buffer ID, automatically assigned to the configured PDUs in the GenTool, must be configured in CANape accordingly. All TX PDUs are given an ascending buffer ID starting with 0. All Rx PDUs are attached behind the TX PDUs.

The following configuration in GENy:

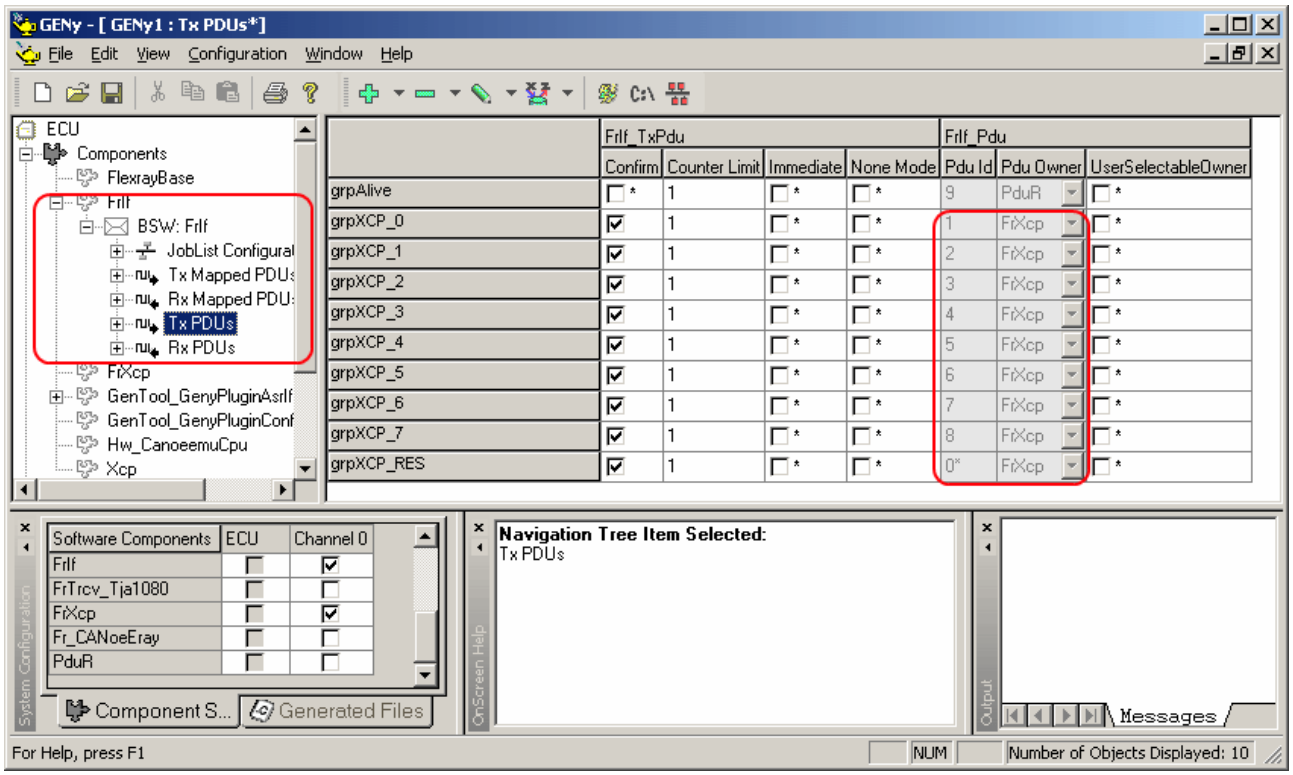

Figure 4-1 CANape – PDU relation

must match the configuration of the buffers in the a2l file with the Pdu Id as buffer number. For easier integration the GenTool generates an FrXcp.a2l fragment which contains the buffer configuration for the Xcp Master Tool.

4.12.3 Resume Mode

To use the resume mode on FlexRay additional FrXcp specific information must be stored in NVM memory. For this an API exists which returns the data to be stored.

[1] Bandwidth penalty
[2] Performance penalty

4.12.3.1 Store FrXcp data to NVM

The following API is called by the FrXcp and must be implemented by the application:

```
void XcpAppl_DaqTlResumeStore(P2CONST(tXcpDaqTl, AUTOMATIC, FRXCP_APPL_DATA) rtConfigPt )
```

The parameter `rtConfigPt` is a pointer to the structure which must be saved. The size of this data can be retrieved by `sizeof(tXcpDaqTl)*(FRXCP_NUM_TX_LPDUIDS + FRXCP_NUM_RX_LPDUIDS)`.

The following API is called when the data has to be cleared from NVM again:

```
void XcpAppl_DaqTlResumeClear( void )
```

4.12.3.2 Restore FrXcp data from NVM

A call of `FrXcp_Init()` will trigger a call of the following call-back:

```
uint8 XcpAppl_DaqTlResume(P2VAR(tXcpDaqTl, AUTOMATIC, FRXCP_APPL_DATA) rtConfigPt )
```

If resume data is available it must be restored in this call back. If no valid resume data is available this function can return doing nothing.

5 Configuration

5.1 A2L File

The GenTool exports an a2l file for easier configuration of the XCP Master (e.g. CANape). This file is called FrXcp.a2l and contains the XCP_ON_FLEXRAY IF_DATA section which can be included by a master template a2l file.

Hint!

The following abstract can be used to include the generated a2l:

```
/begin IF_DATA XCP
  ……..
  /include "FrXcp.a2l"
/end IF_DATA
```

5.2 Manual Configuration

It is also possible to configure XCP on FlexRay with the provided configuration templates. The individual configuration options are described in the following sub-chapters.

5.2.1 Pre-Compile Configuration

The following configuration options can be used to configure XCP on FlexRay during pre-compile time:

Configuration options	Value	Description
kFrXcpMaxCTO	> 12..253	This parameter defines the maximum length of Command Transfer Objects (CTO) in bytes. For FlexRay this parameter must be at least 12 and less than the maximum length of the PDU or Link-Layer frame.
kFrXcpMaxDTO	> 12..253	This parameter defines the maximum length of Data Transfer Objects (DTO). For FlexRay this parameter must be at least 12 and less than the maximum length of the PDU or Link-Layer frame.
FRXCP_PDU_SIZE	> 14..255	This parameter defines the maximum length of the PDU. This parameter must be at least Max. DTO + 2..4, depending on other configuration options (Frame Alignment, Frame Concatenation)

FRXCP_DEV_ERROR_DETECT	■ ON ■ OFF	Activate/Deactivate the development error detection. The development error detection comprises amongst others channel parameter checks and check of initialization upon every service call. It is strongly recommended to disable this option in production code.
FRXCP_USE_DECOUPLED_MODE	■ ON ■ OFF	It is possible to use the XCP either in Immediate or Decoupled Mode. Immediate Mode requires less resources but the XCP Event function must be used synchronous to the FlexRay bus, while in Decoupled Mode sampling is independent of the FlexRay bus.
FRXCP_FRAME_CONCATENATION	■ ON ■ OFF	With this attribute you can enable the concatenation of multiple XCP frames into one Protocol Data Unit (PDU). Be aware of higher protocol overhead and hence lower XCP bandwidth if enabled!
FRXCP_SEQUENCE_COUNTER	■ ON ■ OFF	The Sequence Counter can be used for safety reasons to detect missing frames and frame mix ups. An enabled Sequence Counter requires a slightly higher bandwidth overhead.
FRXCP_USE_BUFFER_RECONFIG_API	■ ON ■ OFF	This parameter enables an API that allows dynamic reconfiguration of physical buffers. The use case is to save physical buffers while maintaining flexibility. For this to work the FlexRay driver must support an enhanced API for buffer reconfiguration.
FRXCP_FRAME_ALIGNMENT_[X]BIT	■ 8 ■ 16 ■ 32	This parameter determines the Alignment of XCP frames within a PDU or Link-Layer frame. This parameter must be selected according to performance and architectural needs.
FRXCP_USE_PDUMODE	■ ON ■ OFF	Can be used to prevent XCP frames to be sent, e.g. if the bus is offline. The frames are stored in the send queue until the bus is available again
FRXCP_CONFIG_VARIANT	■ 1 ■ 2 ■ 3	Configuration variant 1=Pre Compile 2=Link time 3=Post build

Table 5-1 Pre-Compile Configuration

5.2.2 Link Time & Post Build Configuration

The following configuration options can be used to configure XCP on FlexRay during post-build-time.

Configuration options	Value	Description
FrXcp_RootConfig		

FrXcpPduDescriptorList	> Pointer	Pointer to pdu configuration, see below
FrXcp_NAX	> 0u...255u	This unique node address identifies the XCP slave within a network. This address is normally defined by the system designer and can be the same address as used in the Network Management, for example.
ConfTimeoutReload	> 0u..65535u	Tx Confirmation timeout timer reload value in MainFunction cycles. Used in sync loss conditions when the FrIf does not provide TxConfirmations anymore.
FrXcpPduDescriptorListSize	> 1u...255u	Number of PDU descriptors in the following List
NumberOfTransmitFC	> 0u..255u	Number of pdu buffers.

FrXcpNumberOfTxPdus	> 1u...255u	Number of Tx PDUs
FrXcpNumberOfRxPdus	> 1u...255u	Number of Rx PDUs
FrXcpMaxTxPduID	> 1u…255u	Max PDU ID for Tx PDUs
FrXcpMaxRxPduID	> 1u...255u	Max PDU ID for Rx PDUs
> FrXcpPduDescriptorList (for each Tx PDU)		
XcpPduId	> 0u…255u > 0u…65535u	Unique PDU ID of underlying layer, e.g. PDU Router or FlexRay Interface that is used for transmission of messages by the XCP.
XcpPacketFilter	■ LPDU_TYPE_VARIABLE ■ LPDU_TYPE_RES ■ LPDU_TYPE_EV ■ LPDU_TYPE_DAQ	The Frame Type determines the type of XCP frames that can be sent via this PDU.
MaxFlxLenBuf	> 14u...254u	Maximum length of this specific buffer.
IsReconfigurable	> 0; 1	Is this PDU reconfigurable via FLX_ASSIGN
IsInitialized	> 0; 1	If the XcpPacketFilter is not of type Variable this value must be 1.

Table 5-2 Post Build Configuration

6 Description of the API

The following chapter contains a brief description of the API provided by the XCP on FlexRay component.

6.1 Data Types

The software module XCP on FlexRay uses the standard AUTOSAR data types that are defined within `Std_Types.h` and the platform specific data types that are defined within `Platform_Types.h`.

Furthermore the following software module specific data types are used:

Name	Type	Description
XCP on FlexRay		
FrXcpPduDescriptorType	struct	Post-Compile configuration structure

Table 6-1 Data Types

6.2 Global Variables

There are no global variables within XCP on FlexRay.

6.3 Global Constants

6.3.1 Component Versions

The component version of XCP on FlexRay is provided by three BCD constants. These constants are declared as external and can be read by the application at any time.

Constant name	Type	Description Value
XCP on FlexRay		
`FRXCP_MAJOR_VERSION`	BCD	Contains the major version number.
`FRXCP_MINOR_VERSION`	BCD	Contains the minor version number.
`FRXCP_PATCH_VERSION`	BCD	Contains the patch level version number. Patch Version 255 means BETA version

Table 6-2 Version Data

6.3.2 Vendor ID

The Vendor identifier of XCP on FlexRay is provided by the following constant according to HIS:

Constant name	Type	Description Value
XCP on FlexRay		
`FRXCP_VENDOR_ID`	-	Vendor ID according to HIS. Vector Informatik GmbH = 30 (decimal)

Table 6-3 Vendor ID

6.3.3 Module ID

The Module identifier of XCP on FlexRay is provided by the following constant according to HIS:

Constant name	Type	Description Value
XCP on FlexRay		
FRXCP_MODULE_ID	-	Module ID according to HIS. XCP on FlexRay = 211 (decimal)

Table 6-4 Module ID

6.4 Services provided by XCP on FlexRay

6.4.1 Administrative Functions

6.4.1.1 FrXcp_Init: Initialization of XCP on FlexRay

FrXcp_Init

Prototype	
`void FrXcp_Init (P2CONST(FrXcp_ConfigType, AUTOMATIC, FRXCP_PBCFG) CfgPtr)`	
Parameters [in/out/both]	
CfgPtr	Pointer to Post build configuration
Return code	
Void	-
Service ID	
Service ID	0
Functional Description	
Global initialization of the Transport Layer, i.e. all PDUs are set to an initial state.	
Preconditions	
None.	
Postconditions	
The Transport Layer will be initialized to the initial state. No PDUs will be sent, by default	
Particularities and Limitations	
■ Call context: task level ■ Not re-entrant ■ Synchronous ■ Parameter CfgPtr is only evaluated in post build configuration	

6.4.1.2 FrXcp_MainFunctionRx: Main Function of XCP Transport Layer

FrXcp_MainFunctionRx

Prototype	
`void FrXcp_MainFunctionRx ( void )`	
Parameters [in/out/both]	
`None`	-
Return code	
`void`	-
Service ID	
Service ID	200
Functional Description	
This service is responsible for cyclic reception handling of XCP-PDUs. This, together with the FrXcp_MainFunctionTx is the only service of the Transport Layer that has to be called by the application. All other services are called by the Protocol Layer exclusively.	
Preconditions	
The Transport Layer must be initialized.	
Postconditions	
None.	
Particularities and Limitations	
■ Call context: task level ■ Not re-entrant ■ Synchronous	

6.4.1.3 FrXcp_MainFunctionTx: Main Function of XCP Transport Layer

FrXcp_MainFunctionTx

Prototype	
`void FrXcp_MainFunctionTx ( void )`	
Parameters [in/out/both]	
`None`	-
Return code	
`void`	-
Service ID	
Service ID	201
Functional Description	
This service is responsible for cyclic transmission handling of XCP-PDUs. This, together with the FrXcp_MainFunctionRx is the only service of the Transport Layer that has to be called by the application. All other services are called by the Protocol Layer exclusively.	
Preconditions	
The Transport Layer must be initialized.	

Postconditions
None.
Particularities and Limitations
■ Call context: task level ■ Not re-entrant ■ Synchronous

6.4.2 Service Callback Functions

6.4.2.1 FrXcp_TriggerTransmit: Call back for transmission of L-PDU

FrXcp_TriggerTransmit

Prototype	
`void FrXcp_TriggerTransmit( const PduIdType XcpTxPduId, PduInfoType *PduInfoPtr );`	
Parameters [in/out/both]	
`XcpTxPduId [in]`	ID of XCP L-PDU that has been triggered
`SduPtr [out]`	Contains Pointer and Length to triggered XCP L-SDU
Return code	
`void`	-
Service ID	
Service ID	103
Functional Description	
This function is called by the FlexRay Interface when a XCP L-PDU has to be transmitted. It copies the XCP L-SDU with respect to the triggered XCP L-PDU ID.	
Preconditions	
The XCP and the FlexRay Interface are configured for Decoupled Mode. Use PduInfoType is enabled	
Postconditions	
Decoupled Mode: The respective PDU buffer is freed again.	
Particularities and Limitations	
■ Call context: task level or interrupt context ■ Re-entrant ■ Synchronous	

6.4.2.2 FrXcp_TxConfirmation: Call back for transmission confirmation of L-PDU

FrXcp_TxConfirmation

Prototype
`void FrXcp_TxConfirmation( const PduIdType XcpTlTxPduId );`

Parameters [in/out/both]	
`XcpTlTxPduId [in]`	ID of XCP L-PDU that has been triggered
Return code	
`void`	-
Service ID	
Service ID	101
Functional Description	
This function is called by the FlexRay Interface after a XCP L-PDU has been transmitted.	
Preconditions	
None.	
Postconditions	
Immediate Mode: The respective PDU buffer is freed again	
Particularities and Limitations	
■ Call context: task level or interrupt context ■ Re-entrant ■ Synchronous	

6.4.2.3 FrXcp_RxIndication: Call back for reception of L-PDU

FrXcp_RxIndication

Prototype	
`void FrXcp_RxIndication( const PduIdType XcpRxPduId,` `const PduInfoType *PduInfoPtr );`	
Parameters [in/out/both]	
`XcpRxPduId [in]`	ID of XCP L-PDU that has been received
`PduInfoPtr [in]`	Contains Pointer and Length to received XCP L-SDU
Return code	
`void`	-
Service ID	
Service ID	102
Functional Description	
This function is called by the FlexRay Interface after a XCP L-PDU has been received. It copies the received L-SDU and stores it locally with respect to the received XCP L-PDU ID.	
Preconditions	
A valid XCP frame was received Use PduInfoType is enabled	
Postconditions	
None.	

Particularities and Limitations
■ Call context: task level or interrupt context ■ Re-entrant ■ Synchronous

6.4.3 Service Functions

6.4.3.1 FrXcp_TLService: Handles Transport Layer Command

FrXcp_TLService

Prototype	
`void FrXcp_TLService ( const uint8* pCmd );`	
Parameters [in/out/both]	
`pCmd [in]`	Pointer to the transport layer command
Return code	
`uint8`	> Depending on the success of the operation, one of the following return codes is returned: ■ XCP_CMD_DENIED ■ XCP_CMD_OUT_OF_RANGE ■ XCP_CMD_OK ■ XCP_CMD_UNKNOWN
Service ID	
Service ID	201
Functional Description	
This service evaluates a transport layer specific command, performs the required operation and returns the status of the operation. Currently supported transport layer specific commands are: ■ FLX_ASSIGN (partly) ■ FLX_ACTIVATE ■ FLX_DEACTIVATE	
Preconditions	
The XCP Transport Layer must be initialized	
Postconditions	
None.	
Particularities and Limitations	
■ Call context: task level ■ Re-entrant ■ Synchronous	

6.4.3.2 FrXcp_Send: Transmission of CTO or DTO

FrXcp_Send

Prototype	
`uint8 FrXcp_Send ( uint8 len, const uint8 *msg );`	
Parameters [in/out/both]	
`len [in]`	Length of message.
`msg [in]`	A Pointer to the message itself.
Return code	
`uint8`	The service returns `XCP_TP_BUSY` if there is no free PDU buffer available anymore, otherwise XCP_TP_OK is returned.
Service ID	
Service ID	2
Functional Description	
This service prepares transmission of the referenced frame.	
Preconditions	
The XCP Transport Layer must be initialized.	
Postconditions	
-	
Particularities and Limitations	
■ Call context: task level ■ Re-entrant (not with the same channel) ■ Synchronous	

6.4.3.3 FrXcp_SendFlush: Finish pending frames

FrXcp_SendFlush

Prototype	
`void FrXcp_SendFlush ( uint8 XcpFlushTypeSel );`	
Parameters [in/out/both]	
`XcpFlushTypeSel`	Selects which type of xcp frames are flushed. Possible values are: • `XCP_FLUSH_CTO` • `XCP_FLUSH_DTO` • `XCP_FLUSH_ALL`
Return code	
–	–
Service ID	
Service ID	4
Functional Description	
Purge the current Frame Cache. This service is only needed when frame concatenation is enabled.	

Preconditions
The XCP Transport Layer must be initialized.
Postconditions
-
Particularities and Limitations
■ Call context: task level ■ Re-entrant (not with the same channel) ■ Synchronous

6.4.3.4 FrXcp_SetPduMode: Enable/Disable transmission

FrXcp_SetPduMode

Prototype	
`void FrXcp_SetPduMode( NetworkHandleType XcpNwH, FrXcp_PduSetModeType PduMode );`	
Parameters [in/out/both]	
`XcpNwH [in]`	> The network handle is usually 0
`PduMode [in]`	> The PDUMode which can be ■ FRXCP_SET_OFFLINE (default) ■ FRXCP_SET_ONLINE
Return code	
–	–
Service ID	
Service ID	7
Functional Description	
With this service it is possible to prevent transmission of frames. The frames are not lost but stored in the send queue until overrun occurs or transmission is enabled again.	
Preconditions	
> -	
Postconditions	
-	
Particularities and Limitations	
■ Call context: task level ■ Not re-entrant ■ Synchronous	

6.4.3.5 FrXcp_GetVersionInfo: Get Version Information

FrXcp_GetVersionInfo

Prototype
`void FrXcp_GetVersionInfo( Std_VersionInfoType *FrXcpVerInfoPtr );`

Parameters [in/out/both]	
`FrXcpVerInfoPtr [out]`	Pointer to version information structure
Return code	
–	–
Service ID	
Service ID	6
Functional Description	
FrXcp_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Preconditions	
> -	
Postconditions	
-	
Particularities and Limitations	
■ Call context: task level ■ Not re-entrant ■ Synchronous	

6.4.4 Macros

6.4.4.1 XCP_ACTIVATE: Enable the Protocol and Transport Layer

XCP_ACTIVATE

Prototype	
`XCP_ACTIVATE();`	
Parameters [in/out/both]	
–	-
Return code	
–	–
Service ID	
Service ID	-
Functional Description	
With this service it is possible to enable all functionality of the XCP Protocol and Transport Layer during runtime to prevent erroneous execution.	
Preconditions	
> -	
Postconditions	
-	

Particularities and Limitations
■ Call context: task level ■ Not re-entrant ■ Synchronous

6.4.4.2 XCP_DEACTIVATE: Disable the Protocol and Transport Layer

XCP_DEACTIVATE

Prototype	
`XCP_DEACTIVATE();`	
Parameters [in/out/both]	
–	-
Return code	
–	–
Service ID	
Service ID	-
Functional Description	
With this service it is possible to lock all functionality of the XCP Protocol and Transport Layer during runtime to prevent erroneous execution.	
Preconditions	
> -	
Postconditions	
-	
Particularities and Limitations	
■ Call context: task level ■ Not re-entrant ■ Synchronous	

6.5 Services used by XCP on FlexRay

The following table lists services which are provided by other components and used by XCP on FlexRay. For details about prototype and functionality refer to the documentation of the respective component. The services provided by the XCP Protocol Layer are not listed here.

Component	API

XCP Hal	`ApplXcpDaqTlResumeClear`
	`ApplXcpDaqTlResumeStore`
	`ApplXcpDaqTlResume`

FlexRay Interface	`FrIf_Transmit`
XCP PI	`Xcp_TlMainFunction`

	Xcp_SendCallBack
Development Error Tracer[1]	Det_ReportError
AUTOSAR OS	SuspendAllInterrupts ResumeAllInterrupts

Table 6-5 Services used by XCP on FlexRay

6.6 Development Error Tracer

The following table contains the supported DET error codes:

Name	Error Id	Description AUTOSAR / OEM / Vector specific
XCP on FlexRay		
FRXCP_E_NOT_INITIALIZED	0x01	A XCP Transport Layer service was called without initializing the module first by calling FrXcp_Init.
FRXCP_E_INV_PDU_ID	0x02	The Xcp Transport Layer was called with an invalid PDU-ID.
FRXCP_E_NULL_POINTER	0x03	The Xcp Transport Layer was called with a Null pointer as parameter.
FRXCP_E_RX_BUFFER_OVERFLOW	0x04	The Rx MainFunction was not called often enough.
FRXCP_E_RX_INVALID_LENGTH	0x06	The received pdu has an illegal length.

Table 6-6 Development Error Detection Codes of XCP on FlexRay

6.7 Diagnostic Event Manager

The DEM is not used by the XCP.

7 Extensions

The following extensions of the XCP on FlexRay software specifications are available within the FlexRay embedded software components. If required, the extensions have to be enabled during configuration:

None.

8 Known Issues/ Limitations

8.1 Reconfig LPDU

Reconfiguration of LPDUs is currently not supported.

9 Icons

Caution
This symbol calls your attention to warnings.

Info
Here you can obtain supplemental information.

Practical Procedure
Step-by-step instructions provide assistance at these points.

Example
Here is an example that has been prepared for you.

Edit
Instructions on editing files are found at these points.

Do not edit manually
This symbol warns you not to edit the specified file.

FAQ
In this area you can get answers to frequently asked questions.

10 Contact

Visit our website for more information on

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

www.vector-informatik.com