MICROSAR XCP

Technical Reference

Version 6.0.2

Authors	vishr
Status	Released

Document Information

History

Author	Date	Version	Remarks
Vishr	2017-02-13	1.00.00	Initial Version
Vishr	2017-11-14	2.00.00	Added new API Xcp_SetStimMode
Vishr	2018-07-06	2.00.01	Fixed Copy&Paste error in chapter 5.5.2
Vishr	2018-11-13	3.00.00	Added Relative ODT, Absolute DAQ mode Some clarifications
Vishr	2019-03-28	4.00.00	STORY-7952 Support IF Modules on different cores for XCP
Vishr	2020-07-29	5.00.00	SWAT-1093 XCP Event Extension to support dynamic memory SWAT-880 XCP multi frame seed/key support
Vishr	2020-09-15	5.00.01	Minor clarifications
Vishr	2021-04-29	6.00.00	Added complementary channel ESCAN00108827 TechRef erroneously states that the Max DTO is limited to 255.
Vishr	2021-08-21	6.00.01	Improved Howto.
vishr	2021-10-04	6.00.02	Improved Howto.

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_XCP.pdf	2.3.0
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	3.4.1
[3]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	5.2.0
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	ASAM	ASAM_AE_MCD-1-XCP_Protocol-Layer_BS_V1-2-0.pdf	V1.2
[6]	VECTOR	UserManual_Multi-Core.pdf	1.1.3

Scope of the Document

This document describes the features, APIs, and integration of the XCP Protocol Layer.

This document does not cover the XCP Transport Layers for CAN, FlexRay and Ethernet, which are available at Vector Informatik.
Further information about XCP on CAN, FlexRay and Ethernet Transport Layers can be found in their documentation.

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

The XCP Protocol Layer is a hardware independent protocol that can be ported to almost any hardware. Due to there are numerous combinations of micro controllers, compilers and memory models it cannot be guaranteed that it will run properly on any 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.

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

Info
The source code of the XCP Protocol Layer, configuration examples and documentation are available on the Internet at www.vector-informatik.de in a functional restricted form.

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

Contents

1 Introduction ... 10

1.1 Architecture Overview ... 10

2 Functional Description ... 12

2.1 Features ... 12

2.1.1 Deviations ... 12

2.1.2 Additions/ Extensions ... 14

2.2 Initialization ... 14

2.3 States ... 14

2.4 Main Functions ... 15

2.4.1 Xcp_MainFunction ... 15

2.4.2 Xcp_TlMainFunction ... 15

2.5 Block Transfer Communication Model ... 16

2.6 Slave Device Identification ... 16

2.6.1 XCP Station Identifier ... 16

2.6.2 XCP Generic Identification ... 16

2.7 Seed & Key ... 16

2.8 Checksum Calculation ... 17

2.8.1 Custom CRC calculation ... 17

2.9 Memory Access by Application ... 18

2.9.1 Special use case “Type Safe Copy” ... 18

2.10 Memory Read and Write Protection ... 18

2.11 Event Codes ... 19

2.12 Service Request Messages ... 19

2.13 User Defined Command ... 20

2.14 Synchronous Data Transfer ... 20

2.14.1 Synchronous Data Acquisition (DAQ) ... 20

2.14.2 DAQ Timestamp ... 21

2.14.3 Power-Up Data Transfer ... 21

2.14.4 Data Stimulation (STIM) ... 22

2.14.5 Bypassing ... 22

2.14.6 Data Acquisition Plug & Play Mechanisms ... 22

2.14.7 Event Channel Plug & Play Mechanism ... 23

2.14.8 Send Queue ... 23

2.14.9 Data consistency ... 23

2.14.10 16 Bit PID “Relative ODT, Absolute DAQ” ... 24

2.14.11 Complementary Channel ... 24

2.15 The Online Data Calibration Model ... 25

2.15.1 Page Switching ... 25
2.15.2 Page Switching Plug & Play Mechanism ... 26
2.15.3 Calibration Data Page Copying ... 26
2.15.4 Freeze Mode Handling ... 26
2.16 Flash Programming ... 26
2.16.1 Flash Programming by the ECU's Application ... 27
2.16.2 Flash Programming Plug & Play Mechanism ... 27
2.16.3 Flash Programming with a Flash Kernel ... 27
2.17 Multi Core Support ... 28
2.17.1 Type Safe Copy ... 28
2.17.2 DAQ/STIM with Multi Core ... 29
2.18 Timestamps ... 30
2.19 En- / Disabling the XCP module ... 30
2.20 XCP measurement during the post event time ... 30
2.21 Error Handling ... 31
2.21.1 Development Error Reporting ... 31
2.21.2 Production Code Error Reporting ... 32

3 Integration ... 34
3.1 Scope of Delivery ... 34
3.1.1 Static Files ... 34
3.1.2 Templates – user modifiable ... 34
3.1.3 Dynamic Files ... 34
3.1.4 Generated a2l files ... 34
3.2 Critical Sections ... 35
3.2.1 XCP_EXCLUSIVE_AREA_0 ... 35
3.2.2 XCP_EXCLUSIVE_AREA_1 ... 35
3.2.3 XCP_EXCLUSIVE_AREA_2 ... 35
3.3 Memory Mapping ... 36

4 API Description ... 37
4.1 Type Definitions ... 37
4.2 Structure Definitions ... 37
4.3 Services provided by XCP ... 37
4.3.1 Xcp_InitMemory ... 37
4.3.2 Xcp_Init ... 38
4.3.3 Xcp_Event ... 38
4.3.4 Xcp_DaqConfigInit ... 39
4.3.5 Xcp_EventEx ... 40
4.3.6 Xcp_StimEventStatus ... 41
4.3.7 Xcp_MainFunction ... 42

4.3.8 Xcp_SendEvent ... 42
4.3.9 Xcp_PutChar ... 43
4.3.10 Xcp_Print ... 44
4.3.11 Xcp_Disconnect ... 44
4.3.12 Xcp_SendCrm ... 45
4.3.13 Xcp_GetVersionInfo ... 45
4.3.14 Xcp_ModifyProtectionStatus ... 46
4.3.15 Xcp_GetSessionStatus ... 46
4.3.16 Xcp_GetXcpDataPointer ... 47
4.3.17 Xcp_SetStimMode ... 47
4.3.18 Xcp_DaqSetComplementaryChannel ... 49
4.3.19 Xcp_EventGetDaqSize ... 49
4.4 Services provided by the XCP Protocol Layer and called by the XCP Transport Layer ... 50
4.4.1 Xcp_TlMainFunction ... 50
4.4.2 Xcp_TlRxIndication ... 51
4.4.3 Xcp_TlTxConfirmation ... 51
4.4.4 Xcp_SetActiveTl ... 52
4.4.5 Xcp_GetActiveTl ... 52
4.4.6 Xcp_TlQueryMessagePending ... 53
4.5 XCP Transport Layer Services called by the XCP Protocol Layer ... 53
4.5.1 <Bus>Xcp_Send ... 53
4.5.2 <Bus>Xcp_SendFlush ... 54
4.5.3 <Bus>Xcp_TlService ... 55
4.6 Application Services called by the XCP Protocol Layer ... 55
4.6.1 XcpAppl_GetTimestamp ... 56
4.6.2 XcpAppl_GetPointer ... 56
4.6.3 XcpAppl_GetIdData ... 57
4.6.4 XcpAppl_GetSeedLength ... 58
4.6.5 XcpAppl_GetSeed ... 58
4.6.6 XcpAppl_UnlockWriteKey ... 59
4.6.7 XcpAppl_Unlock ... 60
4.6.8 XcpAppl_CalibrationWrite ... 60
4.6.9 XcpAppl_MeasurementRead ... 61
4.6.10 XcpAppl_CheckReadAccess ... 61
4.6.11 XcpAppl_CheckProgramAccess ... 62
4.6.12 XcpAppl_UserService ... 62
4.6.13 XcpAppl_OpenCmdIf ... 63
4.6.14 XcpAppl_SendStall ... 64
4.6.15 XcpAppl_DisableNormalOperation ... 64
4.6.16 XcpAppl_StartBootLoader ... 65

4.6.17 XcpAppl_Reset ... 65
4.6.18 XcpAppl_ProgramStart ... 66
4.6.19 XcpAppl_FlashClear ... 66
4.6.20 XcpAppl_FlashProgram ... 67
4.6.21 XcpAppl_DaqResume ... 68
4.6.22 XcpAppl_DaqResumeStore ... 68
4.6.23 XcpAppl_DaqResumeClear ... 69
4.6.24 XcpAppl_CalResumeStore ... 70
4.6.25 XcpAppl_GetCalPage ... 70
4.6.26 XcpAppl_SetCalPage ... 71
4.6.27 XcpAppl_CopyCalPage ... 72
4.6.28 XcpAppl_SetFreezeMode ... 72
4.6.29 XcpAppl_GetFreezeMode ... 73
4.6.30 XcpAppl_CalculateChecksum ... 73
4.6.31 XcpAppl_ConStateNotification ... 74
4.6.32 XcpAppl_MemCpy ... 75
4.6.33 XcpAppl_DaqPrepare ... 75
4.6.34 XcpAppl_DaqCleanup ... 76
4.6.35 XcpAppl_GetSendQueueRef ... 76
4.6.36 XcpAppl_TransmitSendQueueElement ... 77
4.7 Services used by XCP ... 77

5 How To ... 78
5.1 Configuring multiple channels ... 78
5.2 Communication size parameters ... 78
5.2.1 Max Number of Odt Daq ... 78
5.2.2 Max Number of ODT Entries ... 79
5.2.3 Max Number of Odt Stim ... 79
5.3 Resource Consumption ... 79
5.3.1 ECU RAM Consumption ... 79
5.3.2 CPU Execution Time ... 79

6 Configuration ... 80
6.1 Configuration Variants ... 80

7 Glossary and Abbreviations ... 81
7.1 Abbreviations ... 81

8 Contact ... 83

Illustrations

Figure 1-1 AUTOSAR 4.1 Architecture Overview ... 10
Figure 1-2 Interfaces to adjacent modules of the XCP ... 11
Figure 2-1 Connection State Machine ... 15
Figure 2-2 Data consistency ... 24
Figure 2-3 API sequence of complementary channel. ... 25
Figure 2-4 Application of Xcp_Event function on Multi Core systems ... 29
Figure 5-1 Configuring a Xcp channel. ... 78

Tables

Table 2-1 Supported AUTOSAR standard conform features ... 12
Table 2-2 Deviations from AUTOSAR standard conform features ... 12
Table 2-3 Deviations from ASAM standard conform features ... 14
Table 2-4 Features provided beyond the AUTOSAR standard ... 14
Table 2-5 States ... 14
Table 2-6 Event codes ... 19
Table 2-7 Xcp_DaqConfigType ... 20
Table 2-8 Service IDs ... 32
Table 2-9 Errors reported to DET ... 32
Table 2-10 Errors reported to DEM ... 33
Table 3-1 Static files ... 34
Table 3-2 Templates ... 34
Table 3-3 Generated files ... 34
Table 4-1 Type definitions ... 37
Table 4-2 Structure definitions ... 37
Table 4-3 Xcp_InitMemory ... 38
Table 4-4 Xcp_Init ... 38
Table 4-5 Xcp_Event ... 39
Table 4-6 Xcp_DaqConfigInit ... 40
Table 4-7 Xcp_EventEx ... 41
Table 4-8 Xcp_StimEventStatus ... 42
Table 4-9 Xcp_MainFunction ... 42
Table 4-10 Xcp_SendEvent ... 43
Table 4-11 Xcp_PutChar ... 44
Table 4-12 Xcp_Print ... 44
Table 4-13 Xcp_Disconnect ... 45
Table 4-14 Xcp_SendCrm ... 45
Table 4-15 Xcp_GetVersionInfo ... 46
Table 4-16 Xcp_ModifyProtectionStatus ... 46
Table 4-17 Xcp_GetSessionStatus ... 47
Table 4-18 Xcp_GetXcpDataPointer ... 47
Table 4-19 Xcp_SetStimMode ... 48
Table 4-20 Xcp_DaqSetComplementaryChannel ... 49
Table 4-21 Xcp_EventGetDaqSize ... 50
Table 4-22 Xcp_TlMainFunction ... 50
Table 4-23 Xcp_TlRxIndication ... 51
Table 4-24 Xcp_TlTxConfirmation ... 52
Table 4-25 Xcp_SetActiveTl ... 52
Table 4-26 Xcp_GetActiveTl ... 53
Table 4-27 Xcp_TlQueryMessagePending ... 53
Table 4-28 <Bus>Xcp_Send ... 54

Table 4-29 <Bus>Xcp_SendFlush ... 55
Table 4-30 <Bus>Xcp_TlService ... 55
Table 4-31 XcpAppl_GetTimestamp ... 56
Table 4-32 XcpAppl_GetPointer ... 57
Table 4-33 XcpAppl_GetIdData ... 57
Table 4-34 XcpAppl_GetSeedLength ... 58
Table 4-35 XcpAppl_GetSeed ... 59
Table 4-36 XcpAppl_UnlockWriteKey ... 59
Table 4-37 XcpAppl_Unlock ... 60
Table 4-38 XcpAppl_CalibrationWrite ... 61
Table 4-39 XcpAppl_MeasurementRead ... 61
Table 4-40 XcpAppl_CheckReadAccess ... 62
Table 4-41 XcpAppl_CheckProgramAccess ... 62
Table 4-42 XcpAppl_UserService ... 63
Table 4-43 XcpAppl_OpenCmdIf ... 64
Table 4-44 XcpAppl_SendStall ... 64
Table 4-45 XcpAppl_DisableNormalOperation ... 65
Table 4-46 XcpAppl_StartBootLoader ... 65
Table 4-47 XcpAppl_Reset ... 66
Table 4-48 XcpAppl_ProgramStart ... 66
Table 4-49 XcpAppl_FlashClear ... 67
Table 4-50 XcpAppl_FlashProgram ... 68
Table 4-51 XcpAppl_DaqResume ... 68
Table 4-52 XcpAppl_DaqResumeStore ... 69
Table 4-53 XcpAppl_DaqResumeClear ... 69
Table 4-54 XcpAppl_CalResumeStore ... 70
Table 4-55 XcpAppl_GetCalPage ... 71
Table 4-56 XcpAppl_SetCalPage ... 71
Table 4-57 XcpAppl_CopyCalPage ... 72
Table 4-58 XcpAppl_SetFreezeMode ... 73
Table 4-59 XcpAppl_GetFreezeMode ... 73
Table 4-60 XcpAppl_CalculateChecksum ... 74
Table 4-61 XcpAppl_ConStateNotification ... 74
Table 4-62 XcpAppl_MemCpy ... 75
Table 4-63 XcpAppl_DaqPrepare ... 75
Table 4-64 XcpAppl_DaqCleanup ... 76
Table 4-65 XcpAppl_GetSendQueueRef ... 77
Table 4-66 XcpAppl_TransmitSendQueueElement ... 77
Table 4-67 Services used by the XCP ... 77
Table 5-1 ECU RAM Consumption ... 79
Table 7-1 Abbreviations ... 82

1 Introduction

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

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	XCP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	XCP_MODULE_ID	212 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

1.1 Architecture Overview

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

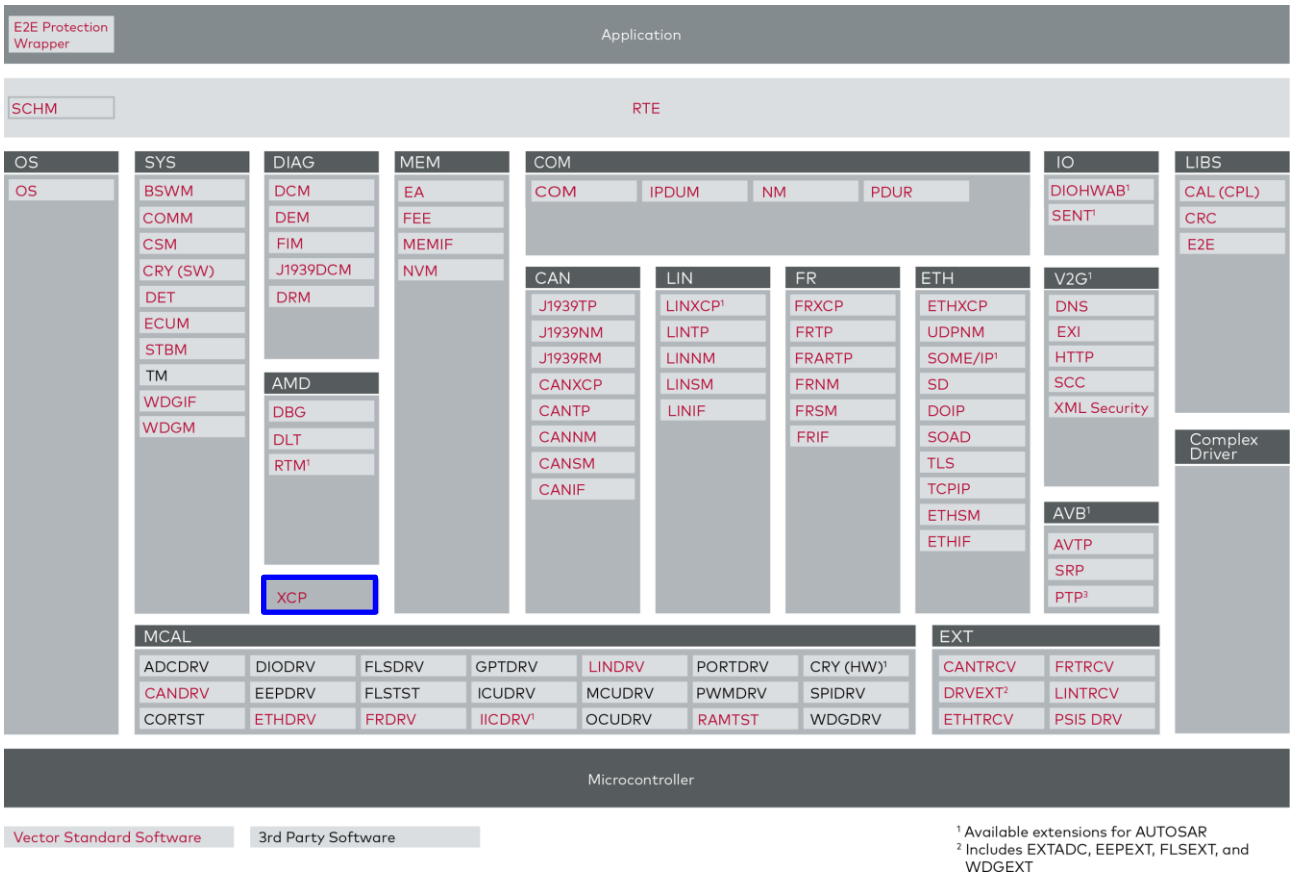


Figure 1-1 AUTOSAR 4.1 Architecture Overview

The following figure shows the interfaces to adjacent modules of the XCP. The interfaces of the XCP Protocol Layer and the application call-back header are described in chapter 4.

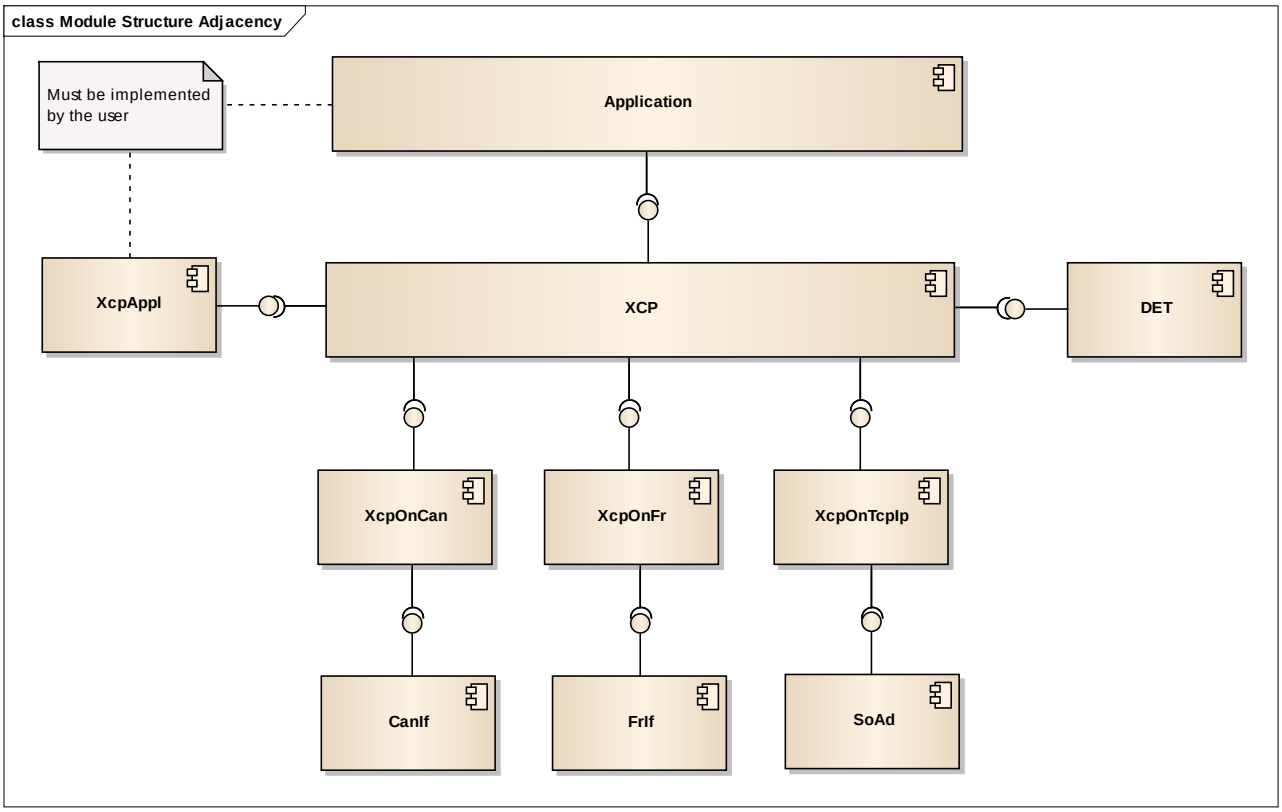


Figure 1-2 Interfaces to adjacent modules of the XCP

2 Functional Description

2.1 Features

The Universal Measurement and Calibration Protocol (XCP) is standardized by the European ASAM working committee for standardization of interfaces used in calibration and measurement data acquisition. XCP is a higher level protocol used for communication between a measurement and calibration system (MCS, i.e. CANape) and an electronic control unit (ECU). The implementation supports the ASAM XCP 1.1 Specification.

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

> Table 2-1 Supported AUTOSAR standard conform features

> Table 2-2 Deviations from AUTOSAR standard conform features

> Table 2-3 Deviations from ASAM standard conform features

Vector Informatik provides further XCP functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 2-4 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
ASAM XCP Version 1.1

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

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

Category	Description	ASR Version
Functional	The following features are not supported: • The command `GET_SLAVE_ID` • A CDD as transport layer	4.2.2
API	The following APIs are not provided by XCP: • Xcp_SetTransmissionMode	4.2.2
API	The API Xcp_<Module>TriggerTransmit is only supported for transport layer FrIf.	4.2.2

Table 2-2 Deviations from AUTOSAR standard conform features

Category	Description	ASAM Version
Functional	1.6.4.1.2.4 Get general information on DAQ processor:	1.1

	• Bitwise stimulation is not supported	
Functional	1.6.4.2 Static DAQ list configuration (stat): • Static DAQ lists are not supported; only dynamic DAQ lists are supported	1.1
Functional	1.7.2.3 Interleaved Communication Model: • Multiple request messages are not allowed to be transmitted by the XCP master before receiving the corresponding response message	1.1
Functional	1.6.5.2.4 Set Data Format before Programming: • Only the default programming format is supported, therefore the command `PROGRAM_FORMAT` is not supported	1.1
Functional	1.6.5.2.2 Get specific information for a sector: • The command `GET_SECTOR_INFO` does not return a Program Sequence Number	1.1
Functional	1.6.5.2.7 Program Verify: • The command `PROGRAM_VERIFY` is not supported	1.1
Functional	Daq configuration: • Number of DAQ lists is limited to 0xFF • DAQ-list and event channel prioritization is not supported • DAQ bit offset not supported • The resume bits in DAQ lists are not set (no indication in response of command `GET_DAQ_LIST_MODE`)	1.1
Functional	5.1.10 ODT Optimization: • The ODT Optimization is not supported	1.2
Functional	1.2 Table of Event Codes: • XCP does not send any event packet natively. If required, the implementation has to be added to application	1.1
Functional	Overload indication by an event is not supported	1.1
Functional	1.3 Table of Service Request Codes (SERV): • The Service Request `SERV_RESET` is not supported	1.1
Functional	1.6.1.2.9 Build Checksum over memory range: • The checksum type `XCP_CRC_16` or `XCP_CRC_32` is only supported if the checksum calculation is forwarded to a AUTOSAR CRC module • Maximum checksum block size is 0xFFFF	1.1
Functional	1.6.3 Page Switching Commands (PAG): • The command `GET_PAGE_INFO` is not supported • The command `GET_SEGMENT_INFO` is not supported • Only one segment and two pages are supported	1.1
Functional	Seed and Key: • The seed size and key size must be equal or less MAX_CTO-2	1.1
Functional	Consistency only supported on ODT level.	1.1
Functional	STIM only supports one DAQ list	1.1

Table 2-3 Deviations from ASAM standard conform features

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Support of CAN-FD
Support transmission and reception of DTO on multiple cores simultaneously.

Table 2-4 Features provided beyond the AUTOSAR standard

2.2 Initialization

The XCP gets initialized by call of the following services:

- 4.3.1 Xcp_InitMemory
- 4.3.2 Xcp_Init

Xcp_InitMemory has to be called if memory is not initialized by start-up code.

The EcuM takes care of initialization, if no EcuM is used these functions have to be called by application in correct order.

2.3 States

The XCP's connection state machine is shown in Figure 2-1, comprises the following states:

State Name	Description
`XCP_CON_STATE_DISCONNECTED`	In this state neither CTO nor DTO messages can be received or transmitted, except of the Connect CTO.
`XCP_CON_STATE_CONNECTED`	In this state communication is fully supported.
`XCP_CON_STATE_RESUME`	In this state CTO messages (except of Connection CTO) are rejected, whereas DTO messages can be received and transmitted.

Table 2-5 States

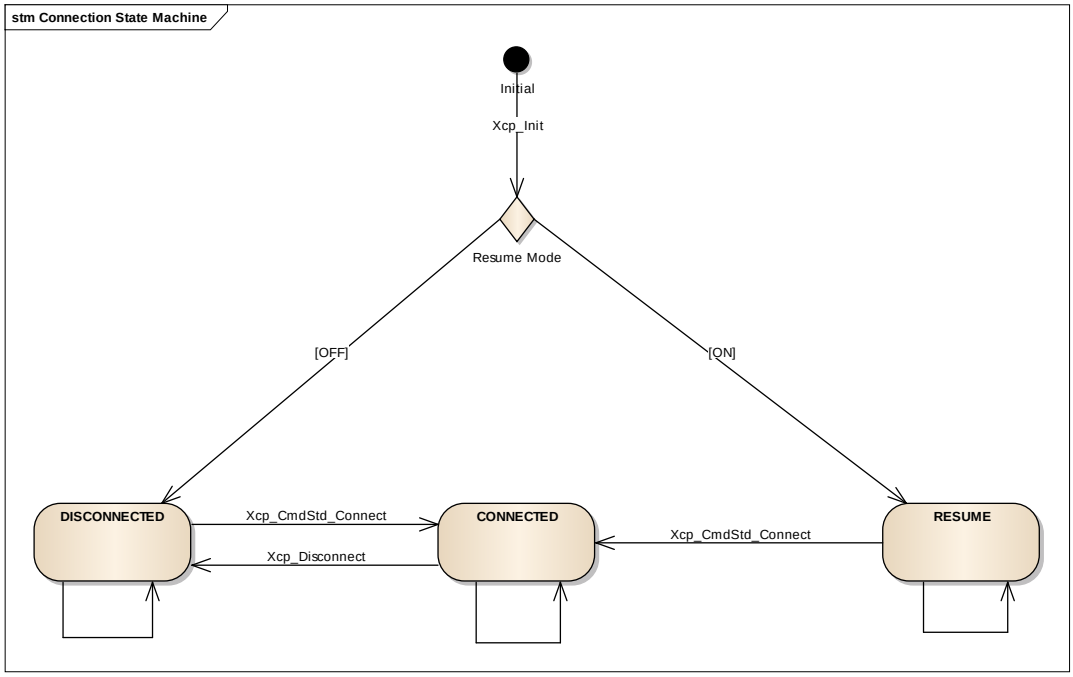


Figure 2-1 Connection State Machine

The states can be changed by the XCP master by sending the CTOs `Connect` and `Disconnect`. Additionally, the connection can be broken by the service:

- 4.3.11 Xcp_Disconnect

2.4 Main Functions

The Xcp provides the following MainFunctions:

- 4.3.7 Xcp_MainFunction
- 4.4.1 Xcp_TlMainFunction

they must be called cyclically and perform the following tasks:

2.4.1 Xcp_MainFunction

> Checksum calculation which is done asynchronously in configurable chunks to prevent extensive runtime

> Resume Mode Handling

The Xcp MainFunction is normally called by the SchM. If you use a 3rd party SchM you must configure it accordingly such that the function is called cyclically.

2.4.2 Xcp_TlMainFunction

> Triggers transmission of DAQ and Event/Serv messages on the respective channel.

The Xcp TlMainFunction is normally called by the respective transport layer for each channel and has the granularity of the respective transport layer MainFunction cycle. This function can also be used to trigger transmission of DAQ frames explicitly.

Caution
If a transport layer is executed on a different core, please keep in mind that the application callbacks are executed in the same context.

2.5 Block Transfer Communication Model

In the standard communication model, each request packet is responded by a single response packet or an error packet. To speed up memory uploads, downloads and flash programming the XCP commands `UPLOAD`, `DOWNLOAD` and `PROGRAM` support a block transfer mode similar to ISO/DIS 15765-2.

In the Master Block Transfer Mode can the master transmit subsequent (up to the maximum block size MAX_BS) request packets to the slave without getting any response in between. The slave responds after transmission of the last request packet of the block.

In Slave Block Transfer Mode, the slave can respond subsequent (there is no limitation) to a request without additional requests in between.

The Block Transfer Mode is limited to a block size of 255 Bytes. On bus systems with a large max CTO (e.g. 254 Bytes) this Mode might be counterproductive and should stay disabled.

2.6 Slave Device Identification

2.6.1 XCP Station Identifier

The XCP station identifier is an ASCII string that identifies the ECU's software program version.

The MCS can interpret this identifier as file name for the ECU database. The ECU developer should change the XCP station identifier with each program change. This will prevent database mix-ups and grant the correct access of measurement and calibration objects from the MCS to the ECU. Another benefit of the usage of the XCP station identifier is the automatic assignment of the correct ECU database at program start of the MCS via the plug & play mechanism. The plug & play mechanism prevents the user from selecting the wrong ECU database.

2.6.2 XCP Generic Identification

The XCP provides a generic mechanism for identification by the `GET_ID` command. For this purpose, a call-back exist which can be implemented by the user to provide the requested information (see 4.6.3 XcpAppl_GetIdData).

2.7 Seed & Key

The seed and key feature allows individual access protection for calibration, flash programming, synchronous data acquisition and data stimulation. The MCS requests a seed (a few data bytes) from the ECU and calculates a key based on a proprietary algorithm and sends it back to the ECU.

If Seed & Key is enabled in the configuration tool the following APIs need to be implemented by the user:

- 4.6.5 XcpAppl_GetSeed
- 4.6.7 XcpAppl_Unlock

The XcpAppl_GetSeed call-back function returns a seed that is transferred to the MCS. The XcpAppl_Unlock call-back function has to verify a received key based on the seed and then return the resource that shall be unlocked.

The protection state can also individually be modified by the application. The following service can be used for this purpose:

- 4.3.14 Xcp_ModifyProtectionStatus

Note
Annotation for the usage of CANape:

The calculation of the key is done in a DLL, which is developed by the ECU manufacturer and which must be located in the EXEC directory of CANape. CANape can access the ECU only if the ECU accepts the key. If the key is not valid, the ECU stays locked.

2.8 Checksum Calculation

The XCP Protocol Layer supports calculation of a checksum over a specific memory range. The XCP Protocol Layer supports all XCP ADD algorithms and the CRC16CCITT checksum calculation algorithm. If the AUTOSAR CRC Module is used also the XCP CRC32 algorithm can be used.

If checksum calculation is enabled the background task has to be called cyclically.

2.8.1 Custom CRC calculation

The Protocol Layer also allows the calculation of the CRC by the application. For this the call-back is called:

- 4.6.30 XcpAppl_CalculateChecksum

This call-back can either calculate the checksum synchronously and return `XCP_CMD_OK` or it can trigger the calculation and return `XCP_CMD_PENDING` for asynchronous calculation of the checksum. In each case the response frame has to be assembled.

2.9 Memory Access by Application

Memory access to measure or to calibrate variables is performed by two call-backs that can be modified by the user to his needs. Please note that these API are only used for polling access by default. DAQ/STIM uses direct memory access out of performance reasons. DAQ/STIM access via these call-backs can be enabled by `/MICROSAR/Xcp/XcpGeneral/XcpDAQMemAccessByApplication`.

The following call-backs are called by the Protocol Layer whenever a memory access is performed:

- 4.6.8 XcpAppl_CalibrationWrite
- 4.6.9 XcpAppl_MeasurementRead

These APIs can be used to perform the memory access synchronously, asynchronously (e.g. for EEPROM access), and they can deny the memory access, depending on the return value.

2.9.1 Special use case “Type Safe Copy”

The above-mentioned APIs will also be used if the feature “Type Safe Copy” is enabled. If this is the case polling as well as DAQ/STIM measurement will use these functions to read/write data. The template code for these functions performs read/write access in an atomic way for basic data types (e.g. uint16 / uint32).

2.10 Memory Read and Write Protection

Memory protection can easily be performed by the two above-mentioned call-backs returning `XCP_ERR_ACCESS_DENIED`.

Additionally, the configuration switch `/MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpMemoryReadProtection` enables the call-back:

- 4.6.10 XcpAppl_CheckReadAccess

This call-back can be used to verify the address and size of the requested data for services like CRC calculation or WRITE_DAQ. The call-back is called before the service is accepted. Its return code can lead to a negative response (e.g. `XCP_ERR_ACCESS_DENIED`).

As Flash programming uses a different memory access mechanism, a different set of call-backs is used.

The configuration switch `/MICROSAR/Xcp/XcpCmdConfig/XcpProgramming/XcpProgrammingWriteProtection` enables the call-back:

- 4.6.11 XcpAppl_CheckProgramAccess

This call-back can be used to check the memory range whenever a flash segment is cleared or programmed.

2.11 Event Codes

The slave device may report events by sending asynchronous event packets (EV), which contain event codes, to the master device. The transmission is not guaranteed due to the fact that these event packets are not acknowledged.

The transmission of event codes is enabled with the configuration switch `/MICROSAR/Xcp/XcpCmdConfig/XcpAsynchMessage/XcpEventCodes`. The transmission is done by the service:

- 4.3.8 Xcp_SendEvent.

The event codes can be found in the following table.

Event	Code	Description
`XCP_EVC_RESUME_MODE`	`0x00`	The slave indicates that it is starting in RESUME mode.
`XCP_EVC_CLEAR_DAQ`	`0x01`	The slave indicates that the DAQ configuration in non-volatile memory has been cleared.
`XCP_EVC_STORE_DAQ`	`0x02`	The slave indicates that the DAQ configuration has been stored into non-volatile memory.
`XCP_EVC_STORE_CAL`	`0x03`	The slave indicates that the calibration data has been stored.
`XCP_EVC_CMD_PENDING`	`0x05`	The slave requests the master to restart the time-out detection.
`XCP_EVC_DAQ_OVERLOAD`	`0x06`	The slave indicates an overload situation when transferring DAQ lists.
`XCP_EVC_SESSION_TERMINATED`	`0x07`	The slave indicates to the master that it autonomously decided to disconnect the current XCP session.
`XCP_EVC_TIME_SYNC`	`0x08`	Transfer of externally triggered timestamp.
`XCP_EVC_STIM_TIMEOUT`	`0x09`	Indication of a STIM timeout.
`XCP_EVC_SLEEP`	`0x0A`	Slave entering SLEEP mode.
`XCP_EVC_WAKE_UP`	`0x0B`	Slave leaving SLEEP mode.
`XCP_EVC_USER`	`0xFE`	User-defined event.
`XCP_EVC_TRANSPORT`	`0xFF`	Transport layer specific event.

Table 2-6 Event codes

2.12 Service Request Messages

The slave device may request some action to be performed by the master device. This is done by the transmission of a Service Request Packet (SERV) that contains the service request code. The transmission of service request packets is asynchronous and not guaranteed because these packets are not acknowledged.

The service request messages can be sent by the following functions:

- 4.3.9 Xcp_PutChar
- 4.3.10 Xcp_Print

2.13 User Defined Command

The XCP Protocol allows having a user defined command with an application specific functionality. The user defined command is enabled by setting `/MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpUserDefinedCommand` and upon reception of the user command the following callback function is called by the XCP command processor:

- 4.6.12 XcpAppl_UserService

2.14 Synchronous Data Transfer

2.14.1 Synchronous Data Acquisition (DAQ)

The synchronous data transfer can be enabled with the container `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim`. In this mode, the MCS configures tables of memory addresses in the XCP Protocol Layer. These tables contain pointers to measurement objects, which have been configured previously for the measurement in the MCS. Each configured table is assigned to an event channel.

The function Xcp_Event needs to be called for each event channel with the corresponding event channel number as parameter. The application has to ensure that Xcp_Event is called with the correct cycle time. Note that the event channel numbers are given by the GenTool by configuring `/MICROSAR/Xcp/XcpConfig/XcpEventChannel`. Symbolic name values for each event channel are generated by the GenTool and can be found in Xcp_Lcfg.h.

The ECU automatically transmits the current value of the measurement objects via messages to the MCS, when the function Xcp_Event is executed in the ECU's code with the corresponding event channel number. This means that the data can be transmitted at any particular point of the ECU code when the data values are valid.

An extended variant of Xcp_Event is the Xcp_EventEx function. It can be used to provide additional parameters used by the DAQ/STIM measurement. The additional parameters are provided by a structure of type `Xcp_DaqConfigType` that needs to be filled out by the user:

Member type	Structure member	Description
`Xcp_AddressPtrType`	`AddressOffset`	An offset added to the address of each measurement value. This can be used for dynamically allocated parameters.
`Xcp_TimestampType`	`ApplTimestamp`	An application provided timestamp.
`boolean`	`UseApplTimestamp`	Should be TRUE when the above timestamp shall be used instead of the system timestamp.

Table 2-7 Xcp_DaqConfigType

The Xcp_EventEx is used in combination with the Xcp_DaqConfigInit function to initialize the structure with safe default parameters.

The data acquisition mode can be used in multiple configurations that are described within the next chapters.

Note
Annotation for the usage of CANape:

It is recommended to enable both data acquisition plug & play mechanisms to detect the DAQ settings.

2.14.2 DAQ Timestamp

There are two methods to generate timestamps for data acquisition signals.

1. By the MCS tool on reception of the message
2. By the ECU (XCP slave)

The time precision of the MCS tool is adequate for the most applications; however, some applications like the monitoring of the OSEK operating system or measurement on FlexRay with an event cycle time smaller than the FlexRay cycle time require higher precision timestamps. In such cases, ECU generated timestamps are recommended.

The timestamp must be implemented in a call-back which returns the current value:

- 4.6.1 XcpAppl_GetTimestamp

There are several possibilities to implement such a timestamp:

> 16bit Counter variable, incremented by software in a fast task (.e.g. 1ms task) for applications where such a resolution is sufficient and returned in the above mentioned call-back.

> 32bit General Purpose Timer of the used µC, configured to a certain repetition rate (e.g. 1µs increment) for applications that require a high resolution of the timestamp and returned in the above mentioned call-back.

The resolution and increment value of this timer must be configured in the configuration tool accordingly.

2.14.3 Power-Up Data Transfer

Power-up data transfer (also called resume mode) allows automatic data transfer (DAQ) of the slave directly after power-up. Automotive applications would e.g. be measurements during cold start.

The slave and the master need to store all the necessary communication parameters for the automatic data transfer after power-up. Therefore, the following functions have to be implemented in the slave.

- 4.6.21 XcpAppl_DaqResume
- 4.6.22 XcpAppl_DaqResumeStore
- 4.6.23 XcpAppl_DaqResumeClear

To use the resume mode the compiler switch `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpResumeMode` need to be enabled.

Keep also in mind that the Xcp_MainFunction has to be called cyclically in order for the resume mode to work. If Resume Mode is enabled by the MCS tool the before mentioned call-back XcpAppl_DaqResumeStore is called by the Xcp_MainFunction.

Note
Annotation for the use of CANape:

Start the resume mode with the menu command Measurement | Start and push the button “Measure offline” on the dialog box.

2.14.4 Data Stimulation (STIM)

Synchronous Data Stimulation is the inverse mode of Synchronous Data Acquisition.

The STIM processor buffers incoming data stimulation packets. When an event occurs (`Xcp_Event` is called), which triggers a DAQ list in data stimulation mode, the buffered data is transferred to the slave device’s memory.

To use data stimulation (STIM) the configuration switch `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpSynchronousDataStimulation` need to be enabled.

With the API Xcp_SetStimMode the mode of the write operation can be selected.

2.14.5 Bypassing

Bypassing can be realized by making use of Synchronous Data Acquisition (DAQ) and Synchronous Data Stimulation (STIM) simultaneously.

State-of-the-art Bypassing also requires the administration of the bypassed functions. This administration needs to be performed in an MCS like e.g. CANape.

Also, the slave should perform plausibility checks on the data it receives through data stimulation. The borders and actions of these checks are set by standard calibration methods. No special XCP commands are needed for this.

2.14.6 Data Acquisition Plug & Play Mechanisms

The XCP Protocol Layer comprises two plug & play mechanisms for data acquisition:

- General information on the DAQ processor
- General information on DAQ processing resolution

The general information on the DAQ processor contains:

- General properties of DAQ lists
- Total number of available DAQ lists and event channels

The general information on the DAQ processing resolution contains:

- Granularity and maximum size of ODT entries for both directions
- Information on the time stamp mode

2.14.7 Event Channel Plug & Play Mechanism

The XCP Protocol Layer supports a plug & play mechanism that allows the MCS to automatically detect the available event channels in the slave. The associated service is enabled by `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpGetDAQEventInfo`.

If this option is enabled the MCS can read the configured Event Channels from the XCP Slave.

2.14.8 Send Queue

The Send Queue is used to store measurement values until they can be transmitted on the bus. The Send Queue size can be configured in the configuration tool. It is defined by the parameter `/MICROSAR/Xcp/XcpConfig/XcpCoreDefinition/XcpSendQueueSize`. Please be aware that in a Multi Core system multiple Send Queues may be configured. Each Core the Xcp_Event function is called on requires its own Send Queue. The sizes may vary, depending on the number of measurement values on each Core. See chapter 2.17 Multi Core Support.

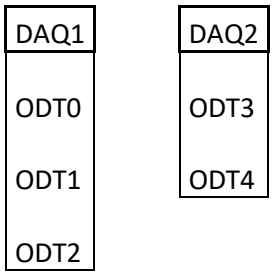
If the Send Queue size is configured too small an overrun condition will happen and measurement values get lost. If the parameter `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpOverrunIndication` is enabled such an overrun condition will get signaled in the MSB of the PID. The Send Queue stores complete ODT messages and needs to be big enough in size to store all messages sampled during an Xcp_Event call and until they can be transmitted via the bus.

2.14.9 Data consistency

The XCP supports a data consistency on ODT level. If a consistency on DAQ level is required, interrupts must be disabled prior calling Xcp_Event and enabled again after the function returns. The following example demonstrates the integrity on ODT level by showing the XCP ODT frames as sent on the bus. Two Events (x, y) are configured with DAQ list DAQ1 assigned to Event(x) and DAQ list DAQ2 assigned to Event(y). A call of the Xcp_Event function with the respective event channel number will then trigger the transmission of the associated DAQ list.

Example1: a call of Xcp_Event(x) is interrupted by a call of Xcp_Event(y). This is allowed as long as the interrupt locks are provided by the Schedule Manager (default with MICROSAR stack).

Example2: a call of Xcp_Event(x) is interrupted by a call of Xcp_Event(x). As a result, a DAQ list is interrupted by itself. This is not allowed and must be prevented by data consistency on DAQ level. For this use a interrupt lock when calling Xcp_Event().



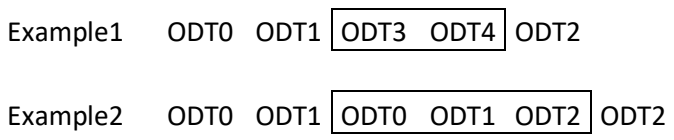


Figure 2-2 Data consistency

Note: For Xcp_EventEx the same limitations apply. It is the same function.

Note: on Multi Core systems: It is in the responsibility of the user to assign only measurement values relevant for the Core to the corresponding Event Channel called on the specific Core.

2.14.10 16 Bit PID “Relative ODT, Absolute DAQ”

The XCP uses as default the “Absolute ODT” as PID. This is an 8bit PID and limits the maximum amount of individual DAQ messages to 123 (if Overrun Indication is enabled).

With the parameter `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpDAQMessageHeader` it is possible to select a 16bit PID “Relative ODT, Absolute DAQ”. This parameter is relevant for huge configurations. It is not recommended to be used in combination with standard CAN as the extended PID will limit bandwidth.

2.14.11 Complementary Channel

The complementary channel is intended for a high performance, high bandwidth, memory mapped channel used to transport DAQ data. This can e.g. be a PCIe interface. The complementary channel is used via a couple of XcpAppl services that handle the memory buffer allocation and management.

The complementary mode can be activated by e.g. a XCP User command. The XCP provides an API to select the respective transmission Mode. The following sequence diagram shows the sequence of APIs:

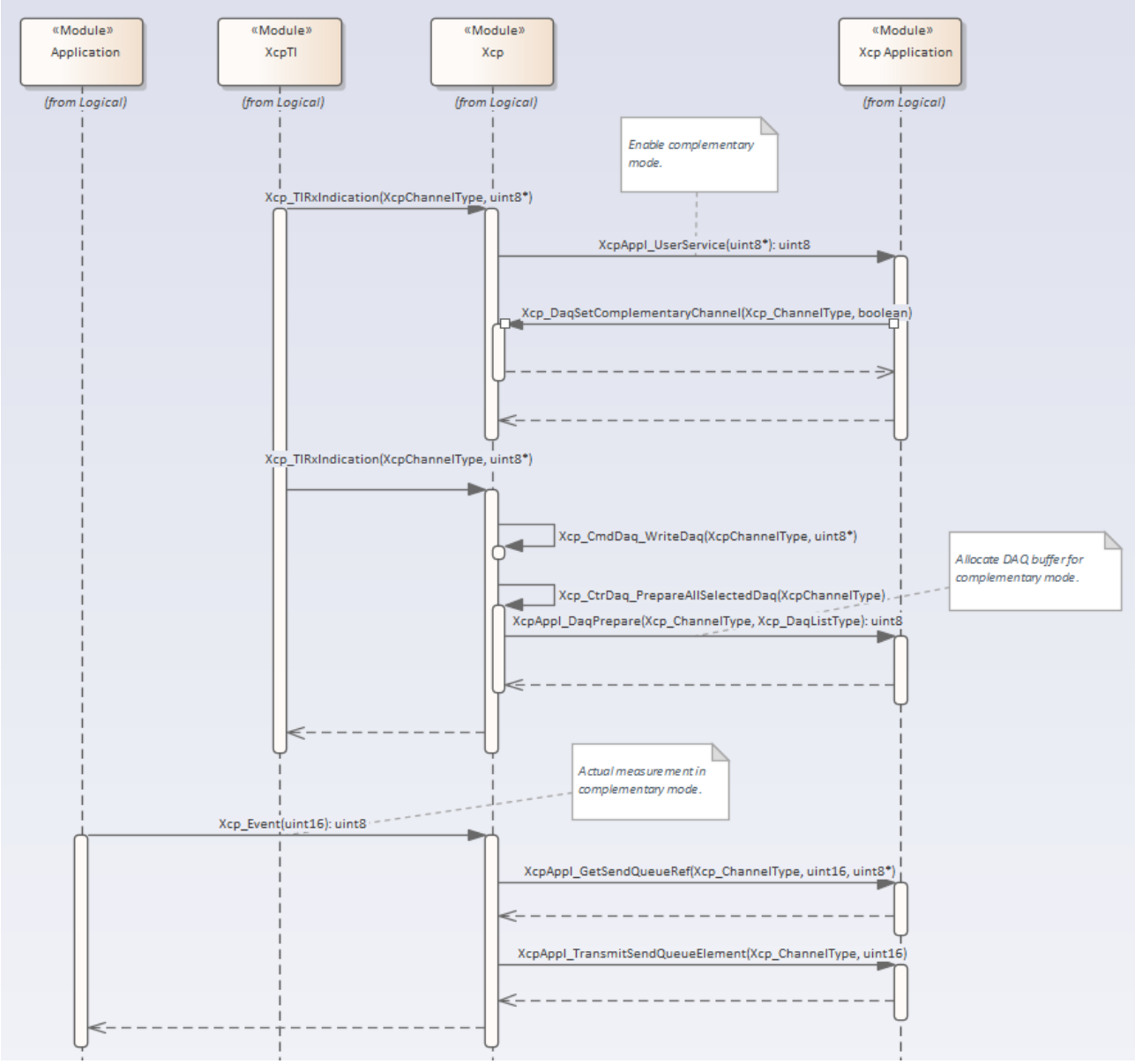


Figure 2-3 API sequence of complementary channel.

2.15 The Online Data Calibration Model

2.15.1 Page Switching

The MCS can switch between a flash page and a RAM page. The XCP command SET_CAL_PAGE is used to activate the required page. The page switching is enabled with the `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching` definition.

The following application callback functions need to be implemented:

- 4.6.25 XcpAppl_GetCalPage
- 4.6.26 XcpAppl_SetCalPage

Note
Annotation for the use of CANape:

Open the dialog XCP Device Setup with the menu command Tools|Driver Configuration. Go to the tab “FLASH”. Activate page switching. Enter a flash selector value e.g. 1 and a Ram selector e.g. 0.

2.15.2 Page Switching Plug & Play Mechanism

The MCS can be automatically configured if the page switching plug & play mechanism is used. This mechanism comprises

> General information about the paging processor

The page switching plug & play mechanism is enabled with the switch `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching/XcpGeneralPagingInfo`.

2.15.3 Calibration Data Page Copying

Calibration data page copying is performed by the XCP command COPY_CAL_PAGE. To enable this feature the compiler switch `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching/XcpCopyPage` has to be enabled.

For calibration data page copying the following application callback function need to be provided by the application:

- 4.6.27 XcpAppl_CopyCalPage

2.15.4 Freeze Mode Handling

Freeze mode handling is performed by the XCP commands SET_SEGMENT_MODE and GET_SEGMENT_MODE. To enable this feature the parameter `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching/XcpFreezeMode` has to be enabled.

For freeze mode handling the following application callback functions have to be provided by the application:

- 4.6.28 XcpAppl_SetFreezeMode
- 4.6.29 XcpAppl_GetFreezeMode
- 4.6.24 XcpAppl_CalResumeStore

2.16 Flash Programming

There are two methods available for the programming of flash memory.

> Flash programming by the ECU’s application

> Flash programming with a flash kernel

Depending on the hardware it might not be possible to reprogram an internal flash sector, while a program is running from another sector. In this case the usage of a special flash kernel is necessary.

2.16.1 Flash Programming by the ECU's Application

If the internal flash has to be reprogrammed and the microcontroller allows to simultaneously reprogram and execute code from the flash the programming can be performed with the ECU's application that contains the XCP. This method is also used for the programming of external flash.

The flash programming is done with the following XCP commands `PROGRAM_START`, `PROGRAM_RESET`, `PROGRAM_CLEAR`, `PROGRAM`, `PROGRAM_NEXT`, `PROGRAM_MAX`, `PROGRAM_RESET`, `PROGRAM_FORMAT`[1], `PROGRAM_VERIFY`[1].

The flash prepare, flash program and the clear routines are platform dependent and therefore have to be implemented by the application.

- 4.6.17 XcpAppl_Reset
- 4.6.18 XcpAppl_ProgramStart
- 4.6.19 XcpAppl_FlashClear
- 4.6.20 XcpAppl_FlashProgram

The flash programming is enabled with the switch `/MICROSAR/Xcp/XcpCmdConfig/XcpProgramming`.

Note
Annotation for the usage of CANape:

Open the dialog XCP Device Setup with the menu command Tools|Driver Configuration. Go to the tab "FLASH" and select the entry "Direct" in the flash kernel drop down list.

2.16.2 Flash Programming Plug & Play Mechanism

The MCS (like e.g. CANape) can get information about the Flash and the Flash programming process from the ECU. The following information is provided by the ECU:

> Number of sectors, start address or length of each sector

> The program sequence number, clear sequence number and programming method

> Additional information about compression, encryption

The flash programming plug & play mechanism is enabled with the switch `/MICROSAR/Xcp/XcpCmdConfig/XcpProgramming/XcpSector`.

2.16.3 Flash Programming with a Flash Kernel

A flash kernel has to be used for the flash programming if it is not possible to simultaneously reprogram and execute code from the flash. Even though the reprogrammed sector and the sector the code is executed from are different sectors.

The application callback function

[1] Command not supported

- 4.6.15 XcpAppl_DisableNormalOperation
- 4.6.16 XcpAppl_StartBootLoader

is called prior to the flash kernel download in the RAM. Within this function the normal operation of the ECU has to be stopped and the flash kernel download can be prepared. Due to the flash kernel is downloaded in the RAM typically data gets lost and no more normal operation of the ECU is possible.

The flash programming with a flash kernel is enabled with the switch `/MICROSAR/Xcp/XcpGeneral/XcpBootloaderDownload`.

Note

Annotation for the usage of CANape:

The flash kernel is loaded by CANape into the microcontroller's RAM via XCP whenever the flash memory has to be reprogrammed. The flash kernel contains the necessary flash routines, its own CAN-Driver and XCP Protocol implementation to communicate via the CAN interface with CANape.

Every flash kernel must be customized to the microcontroller and the flash type being used. CANape already includes some flash kernels for several microcontrollers. There is also an application note available by Vector Informatik GmbH that describes the development of a proprietary flash kernel.

Open the dialog XCP Device Setup with the menu command Tools|Driver Configuration. Go to the tab "FLASH" and select in the 'flash kernel' drop down list, the corresponding *fkl* file for the microcontroller being used.

2.17 Multi Core Support

2.17.1 Type Safe Copy

The XCP Protocol Layer supports a feature called "Type Safe Copy" which provides atomic access to aligned uint16 and uint32 measurement values. This is important on multi core platforms where one core is accessing a measurement value while the XCP is trying to do the same running from another core. The Type Safe Copy is used for polling while DAQ/STIM usually use direct memory access and copy byte wise.

With this option disabled, all access to measurement values is performed byte wise which is not an atomic operation.

The following points must be taken into consideration when enabling this option:

> This option allows the XCP to only read/write basic data types used on another core; it cannot provide data consistency on ODT level.

> This option has a slightly higher runtime.

> Some MCS tools perform an optimization by grouping measurement values. This option must be disabled; otherwise they do not represent unique data types anymore.

2.17.2 DAQ/STIM with Multi Core

It is possible to execute the Xcp_Event and Xcp_EventEx function on a different Core. This must be configured in the configuration tool accordingly. For each Core the XCP is used on the following Container must be created: `/MICROSAR/Xcp/XcpConfig/XcpCoreDefinition`. The correct Core Definition must be referenced for each configured Event Channel: `/MICROSAR/Xcp/XcpConfig/XcpEventChannel/XcpEventChannelCoreRef`. An Event Channel can only be called on the Core it is configured for; otherwise a DET error is thrown.

The following picture shows the architecture behind the Multi Core support and the way the Xcp_Event function is called on each Core:

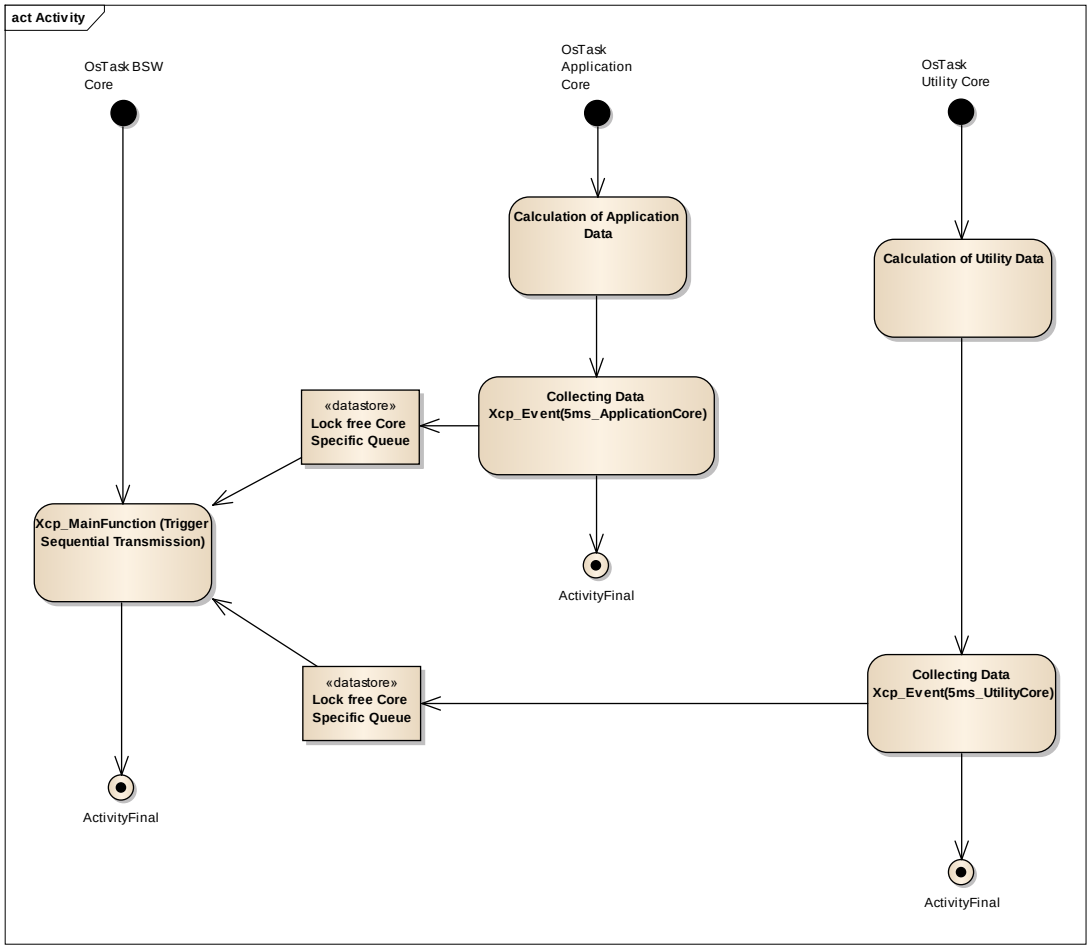


Figure 2-4 Application of Xcp_Event function on Multi Core systems

A detail that needs to be considered is the fact that transmission of DAQ messages is triggered by the <Bus>Xcp_MainFunction. This can be relevant if the MainFunction cycle is relatively slow. In this case transmission might be delayed and then happen in a burst. This may cause incorrect display of the measurement values in CANape regarding their accuracy of chronology. Slave timestamps solve this drawback.

Workaround
Configure and use Slave Timestamps if granularity of bus time stamps is not detailed enough.

2.18 Timestamps

As indicated in the previous chapter, there might be situations when Timestamps are required. The following cases require usage of timestamps:

- Resolution of bus message timestamps is not high enough due to delayed transmission.
- The transport layer (e.g. FlexRay Xcp or Ethernet Xcp) use frame concatenation, e.g. place multiple XCP frame in one bus frame.

If timestamps are enabled, they can be used in two fashions:

1. Determined via the call-out XcpAppl_GetTimestamp. This is the default option. Please note that this call-out is called with active interrupt locks.
2. Determined manually with the extended Xcp_EventEx service. It allows the application to provide the timestamp.

2.19 En- / Disabling the XCP module

The XCP module provides a feature to enable/disable the functionality during runtime by the application. This feature is activated by the option `/MICROSAR/Xcp/XcpGeneral/XcpControl`.

If enabled the macros `XCP_ACTIVATE`/`XCP_DEACTIVATE` can be used to enable or disable the functionality. This can be used to explicitly enable XCP functionality by diagnostic service using its authentification.

The macros control the protocol and transport layer together, i.e. enable or disable them as a whole. It is recommended to perform a Xcp_Disconnect() API call to bring the XCP in a save state before it is being disabled.

This feature is required in ASIL-D use case and described in the Safety Manual how to apply.

2.20 XCP measurement during the post event time

In use cases where there is no further communication request except XCP measurement the session state of the XCP can be determined to prevent an early shutdown of the ECU. For this purpose, the following API exist:

- 4.3.15 Xcp_GetSessionStatus

An example implementation that is called cyclically could look like the following example:

Example

```
{
  uint16 sessionState;
```

```
  sessionState = Xcp_GetSessionStatus();
  if( 0 != (sessionState & XCP_SESSION_CONNECTED) )
  {
    /* Is the xcp actively used? */
    if( 0 != (sessionState & (XCP_SESSION_DAQ | XCP_SESSION_POLLING)) )
    {
      /* Yes, reload timer */
      swTimer = XCPAPPL_TIMEOUT_TIMER_RELOAD;
    }
  }

  if( swTimer > 0 )
  {
    /* No timeout so far */
    swTimer--;
  }
  else
  {
    /* Timer timeout happened, release xcp communication request */
  }
}
```

Please note that polling requests may happen erratically. Therefore, it is important not to choose the timeout value `XCP_TIMEOUT_TIMER_RELOAD` too small.

2.21 Error Handling

2.21.1 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: `/MICROSAR/Xcp/XcpGeneral/XcpDevErrorDetect`.

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 XCP ID is 212.

The reported service IDs identify the services which are described in 4.3. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Xcp_Init
0x03	Xcp_SendEvent
0x04	Xcp_PutChar
0x05	Xcp_Print
0x06	Xcp_Disconnect
0x07	Xcp_SendCrm
0x08	Xcp_GetXcpDataPointer

Service ID	Service
0x0A	Xcp_GetVersionInfo
0x0B	Xcp_TlRxIndication
0x0C	Xcp_TlTxConfirmation
0x0D	Xcp_TlQueryMessagePending
0x0E	Xcp_GetSessionStatus
0x0F	Xcp_SetActiveTl
0x10	Xcp_GetActiveTl
0x11	Xcp_SetStimMode
0x14	Xcp_ModifyProtectionStatus
0x15	Xcp_DaqConfigInit
0x16	Xcp_DaqSetComplementaryChannel
0x17	Xcp_EventGetDaqSize
-	Xcp_MainFunction
-	Xcp_TlMainFunction
0xC9	Xcp_Event
0xFD	Xcp_StimEventStatus

Table 2-8 Service IDs

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

Error Code	Description
0x0A	API service Xcp_Init() called with wrong parameter.
0x0B	API service used with an invalid channel identifier or channel was not configured for the functionality of the calling API.
0x0C	API service used with an invalid event channel identifier or event channel was not configured for the functionality of the calling API.
0x0D	API service used with invalid pointer parameter (NULL).
0x0E	API service used with an invalid channel identifier or channel was not configured for the functionality of the calling API.
0x10	API service used without module initialization.
0x11	The service Xcp_Init() is called while the module is already initialized.
0x12	The service Xcp_Event() is called with a wrong channel id on a wrong core.

Table 2-9 Errors reported to DET

2.21.2 Production Code Error Reporting

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

Error Code	Description
-	No production errors are reported by the XCP.

Table 2-10 Errors reported to DEM

3 Integration

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

3.1 Scope of Delivery

The delivery of the XCP contains the files which are described in the chapters 3.1.1 and 3.1.3:

3.1.1 Static Files

File Name	Description
Xcp.c	This is the source file of the XCP. It contains the XCP protocol layer.
Xcp.h	This is the header file. It contains global declarations.
Xcp_Priv.h	This is the private header file. It contains declarations only relevant for the XCP itself.
Xcp_Types.h	This is the type definition header file. It contains type definitions used by the XCP.

Table 3-1 Static files

3.1.2 Templates – user modifiable

File Name	Description
XcpAppl.c	This is the source file of the application call-back. This file usually must be modified by the user to his needs.
XcpAppl.h	This is the header file of the application call-backs. It contains global declarations.

Table 3-2 Templates

3.1.3 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
Xcp_Cfg.h	XCP Protocol Layer configuration file.
Xcp_Lcfg.c	Parameter definition for the XCP Protocol Layer.
Xcp_Lcfg.h	External declarations for the parameters.

Table 3-3 Generated files

3.1.4 Generated a2l files

The GenTool also generates multiple a2l files which can be used in the MCS tool for easier integration. The following files are generated:

- XCP.a2l (general protocol layer settings)
- XCP_daq.a2l (DAQ specific settings)
- XCP_events.a2l (DAQ event info)

- XCP_Checksum.a2l (Checksum information)

Example Master.a2l:

```
...
/begin IF_DATA XCP
  /include XCP.a2l
  /begin DAQ
    /include XCP_daq.a2l
    /include XCP_events.a2l
    /include XCP_checksum.a2l
    ...
  /end DAQ
  /include CanXCPAsr.a2l
/end IF_DATA
...
/include bsw.a2l
...
```

3.2 Critical Sections

The XCP protocol layer makes use of three critical sections to protect functions that are not re-entrant. The following sections are used:

- `XCP_EXCLUSIVE_AREA_0`
- `XCP_EXCLUSIVE_AREA_1`
- `XCP_EXCLUSIVE_AREA_2`

The individual exclusive areas must not be allowed to interrupt each other. The areas are used for the following cases:

3.2.1 XCP_EXCLUSIVE_AREA_0

This exclusive area is used to protect non-reentrant functions. This critical section covers calls to several sub-functions and can have a long run-time.

3.2.2 XCP_EXCLUSIVE_AREA_1

This exclusive area is used by Xcp_Event during DAQ measurement. It is used to provide data integrity on ODT level and its duration is dependent on the MAX_DTO parameter, i.e. can be short on CAN and long on Ethernet.

3.2.3 XCP_EXCLUSIVE_AREA_2

This exclusive area is used by Xcp_Event during STIM measurement. It is used to provide data integrity on ODT level and its duration is dependent on the MAX_DTO parameter, i.e. can be short on CAN and long on Ethernet.

3.3 Memory Mapping

The XCP has requirements regarding memory mapping to avoid misaligned memory access. The following section: `XCP_START_SEC_VAR_NOCACHE_NOINIT_32BIT` must be mapped to a 32Bit section in order to guarantee correct alignment.

Caution
If this section is not mapped accordingly, a trap will happen on architectures that do not support misaligned access, e.g. TriCore.

4 API Description

For an interface overview please see Figure 1-2.

4.1 Type Definitions

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

Type Name	C-Type	Description
`Xcp_TimestampType`	c-type	This is a type used for timestamp values. Its size is depending on the configuration in the tool and can be uint8, uint16 or uint32.
`Xcp_AddressPtrType`	c-type	XCP internal address type.

Table 4-1 Type definitions

4.2 Structure Definitions

Struct Element Name	C-Type	Description
`Xcp_ChannelStruct`	c-type	This is a complex structure containing all the configuration data of the XCP. This structure needs to be stored in NVM for resume mode.
`Xcp_DaqConfigType`	c-type	This structure contains the extended configuration options for DAQ measurement. Detail can be found in Synchronous Data Acquisition (DAQ).

Table 4-2 Structure definitions

4.3 Services provided by XCP

4.3.1 Xcp_InitMemory

Prototype	
`void` **`Xcp_InitMemory`** `( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
This service initializes the XCP Protocol Layer memory. It must be called from the application program before any other XCP function is called. This is only required if the Startup Code does not initialize the memory with zero.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The global interrupts have to be disabled while this service function is executed. This function should be called during initialization of the ECU before the interrupts have been enabled.
Expected Caller Context
> Task and interrupt level

Table 4-3 Xcp_InitMemory

4.3.2 Xcp_Init

Prototype	
`void Xcp_Init ( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
This service initializes the XCP Protocol Layer and its internal variables. It must be called from the application program before any other XCP function is called (except of Xcp_InitMemory).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Task level	

Table 4-4 Xcp_Init

4.3.3 Xcp_Event

Prototype	
`uint8 Xcp_Event ( uint16 EventChannel )`	
Parameter	
`EventChannel`	Index of event channel to process. The event channel numbers have to start at 0 and have to be continuous. The range is: 0..x. A symbolic name value for each event channel can be found in Xcp_Lcfg.h

Return code	
uint8	XCP_EVENT_NOP : Inactive (DAQ not running, Event not configured) XCP_EVENT_DAQ : DAQ active */ XCP_EVENT_DAQ_OVERRUN : DAQ queue overflow, data lost XCP_EVENT_STIM : STIM active XCP_EVENT_STIM_OVERRUN : STIM data not available

Functional Description

Calling `Xcp_Event` with a particular event channel number triggers the sampling and transmission of all DAQ lists that are assigned to this event channel.

The event channels are defined by the ECU developer in the application program. An MCS (e.g. CANape) must know about the meaning of the event channel numbers. These are usually described in the tool configuration files or in the interface specific part of the ASAM MC2 (ASAP2) database.

Example:

A motor control unit may have a 10ms, a 100ms and a crank synchronous event channel. In this case, the three `Xcp_Event` calls have to be placed at the appropriate locations in the ECU's program:

```
Xcp_Event (XcpConf_XcpEventChannel_10ms); /* 10ms cycle */
xcp_Event (XcpConf_XcpEventChannel_100ms); /* 100ms cycle */
xcp_Event (XcpConf_XcpEventChannel_Crank); /* Crank synchronous cycle */
```

Particularities and Limitations

- Service ID: see table 'Service IDs'
- This function is synchronous.
- This function is reentrant (for different Event Channel).
- The XCP Protocol Layer has been initialized correctly and XCP is in connected state.
- Data acquisition has to be enabled

  /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim

Expected Caller Context

- Task and interrupt level

Table 4-5 Xcp_Event

4.3.4 Xcp_DaqConfigInit

Prototype

void **`Xcp_DaqConfigInit`** (`Xcp_DaqConfigType *pDaqConfig`)

Parameter	
pDaqConfig	Pointer to an empty configuration structure. The configuration structure is initialized with safe default values and can be overwritten by the application afterwards.

Return code	
–	

Functional Description

This function is called to initialize the config structure used as parameter for the Xcp_EventEx function. A typical sequence could look like this:
`{`
`  Xcp_DaqConfigType DaqConfig;`
`  Xcp_DaqConfigInit(&DaqConfig);`
`  DaqConfig.AddressOffset = 1234u;`
`  Xcp_EventEx(XcpConf_XcpEventChannel_10ms, &DaqConfig );`
`}`

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Data acquisition has to be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim
Expected Caller Context
> Task and interrupt level

Table 4-6 Xcp_DaqConfigInit

4.3.5 Xcp_EventEx

Prototype	
`uint8 Xcp_EventEx ( uint16 EventChannel, Xcp_DaqConfigType *pDaqConfig )`	
Parameter	
`EventChannel`	Index of event channel to process. The event channel numbers have to start at 0 and have to be continuous. The range is: 0..x. A symbolic name value for each event channel can be found in Xcp_Lcfg.h
`pDaqConfig`	Pointer to configuration structure containing advanced information about DAQ measurement
Return code	
`uint8`	XCP_EVENT_NOP : Inactive (DAQ not running, Event not configured) XCP_EVENT_DAQ : DAQ active XCP_EVENT_DAQ_OVERRUN : DAQ queue overflow, data lost XCP_EVENT_STIM : STIM active XCP_EVENT_STIM_OVERRUN : STIM data not available

Functional Description

Calling `Xcp_EventEx` with a particular event channel number triggers the sampling and transmission of all DAQ lists that are assigned to this event channel.
The event channels are defined by the ECU developer in the application program. An MCS (e.g. CANape) must know about the meaning of the event channel numbers. These are usually described in the tool configuration files or in the interface specific part of the ASAM MC2 (ASAP2) database.
Example:
A motor control unit may have a 10ms, a 100ms and a crank synchronous event channel. In this case, the three `Xcp_EventEx` calls have to be placed at the appropriate locations in the ECU's program:
`Xcp_EventEx (XcpConf_XcpEventChannel_10ms, &DaqConfig); /* 10ms cycle */`
`xcp_EventEx (XcpConf_XcpEventChannel_100ms, &DaqConfig); /* 100ms cycle */`
`xcp_EventEx (XcpConf_XcpEventChannel_Crank, &DaqConfig); /* Crank synchronous cycle */`
Additionally, this function has a Xcp_DaqConfigType parameter which allows to add an offset to the address of the parameters. This is required for objects located on dynamically allocated memory (e.g. heap variables).

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant (for different Event Channel). > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Data acquisition has to be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim
Expected Caller Context
> Task and interrupt level

Table 4-7 Xcp_EventEx

4.3.6 Xcp_StimEventStatus

Prototype	
`uint8` **`Xcp_StimEventStatus`** `( uint16 EventChannel, uint8 Action )`	
Parameter	
`EventChannel`	Event channel number.
`Action`	`STIM_CHECK_ODT_BUFFER` : check ODT buffer `STIM_RESET_ODT_BUFFER` : reset ODT buffer
Return code	
`uint8`	`XCP_NO_STIM_DATA_AVAILABLE` : stimulation data not available `XCP_STIM_DATA_AVAILABLE` : new stimulation data is available
Functional Description	
Check if data stimulation (STIM) event can perform or delete the buffers.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Data acquisition has to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpSynchronousDataStimulation
Expected Caller Context
> Task and interrupt level

Table 4-8 Xcp_StimEventStatus

4.3.7 Xcp_MainFunction

Prototype	
void **Xcp_MainFunction** (void)	
Parameter	
–	-
Return code	
–	-
Functional Description	
If the XCP command for the calculation of the memory checksum has to be used for large memory areas, it might not be appropriate to block the processor for a long period of time. Therefore, the checksum calculation is divided into smaller sections that are handled in the `Xcp_MainFunction`. Additionally, the main function handles persisting requests.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has been initialized correctly	
Expected Caller Context	
> Task level	

Table 4-9 Xcp_MainFunction

4.3.8 Xcp_SendEvent

Prototype	
Std_ReturnType **Xcp_SendEvent** (Xcp_ChannelType XcpChannel, uint8 EventCode, uint8 *EventData, uint8 Length)	
Parameter	
XcpChannel	The channel number in multi client mode.

EventCode	The event code of the message to send.
EventData	A pointer to the string of the event to send.
Length	The length of the event data.
Return code	
Std_ReturnType	E_OK : Event transmission accepted E_NOT_OK : Event transmission rejected, should be repeated.
Functional Description	
Transmission of event codes via event packets (EV).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Event Codes has to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpAsynchMessage/XcpEventCodes	
Expected Caller Context	
> Task level	

Table 4-10 Xcp_SendEvent

4.3.9 Xcp_PutChar

Prototype	
void **Xcp_PutChar** (Xcp_ChannelType XcpChannel, uint8 *Character)	
Parameter	
XcpChannel	The channel number in multi client mode.
Character	The char to send.
Return code	
–	-
Functional Description	
Put a char into a service request packet (SERV). The service request packet is transmitted if either the maximum packet length is reached (the service request message packet is full) or the character 0x00 is in the service request packet.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Service Request Message has to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpAsynchMessage/XcpServiceRequestMessage	
Expected Caller Context	
> Task level	

Table 4-11 Xcp_PutChar

4.3.10 Xcp_Print

Prototype	
`void Xcp_Print ( Xcp_ChannelType XcpChannel, uint8 *Str )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`Str`	The 0 terminated string to send.
Return code	
–	-
Functional Description	
Transmission of a service request packet (SERV). The string `str` is sent via service request packets. The string has to be terminated by 0x00.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Service Request Message has to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpAsynchMessage/XcpServiceRequestMessage	
Expected Caller Context	
> Task level	

Table 4-12 Xcp_Print

4.3.11 Xcp_Disconnect

Prototype	
`void Xcp_Disconnect ( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
–	-
Functional Description	
If the XCP slave is connected to a XCP master a call of this function discontinues the connection (transition to disconnected state). If the XCP slave is not connected this function performs no action.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The XCP Protocol Layer has been initialized correctly and XCP is in connected state.	

Expected Caller Context
> Task level

Table 4-13 Xcp_Disconnect

4.3.12 Xcp_SendCrm

Prototype	
`void Xcp_SendCrm ( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
–	-
Functional Description	
Transmission of a command response packet (RES), or error packet (ERR) if no other packet is pending.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has been initialized correctly, XCP is in connected state and a command packet (CMD) has been received.	
Expected Caller Context	
> Task level	

Table 4-14 Xcp_SendCrm

4.3.13 Xcp_GetVersionInfo

Prototype	
`void Xcp_GetVersionInfo ( Std_VersionInfoType *versionInfo )`	
Parameter	
`versionInfo`	Pointer to the location where the Version information shall be stored.
Return code	
–	-
Functional Description	
Xcp_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The version info API has to be enabled: /MICROSAR/Xcp/XcpGeneral/XcpVersionInfoApi	

Expected Caller Context
> Task level

Table 4-15 Xcp_GetVersionInfo

4.3.14 Xcp_ModifyProtectionStatus

Prototype	
`void Xcp_ModifyProtectionStatus ( Xcp_ChannelType XcpChannel, uint8 AndState, uint8 OrState )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`AndState`	The following flags: XCP_RM_CAL_PAG, XCP_RM_DAQ, XCP_RM_STIM and XCP_RM_PGM can be used to clear the protection state of the respective resource. The modified state is persistent until Xcp_Init.
`OrState`	The following flags: XCP_RM_CAL_PAG, XCP_RM_DAQ, XCP_RM_STIM and XCP_RM_PGM can be used to set the protection state of the respective resource. The modified state is persistent until Xcp_Init.
Return code	
–	-
Functional Description	
This method can be used to enable or disable the protection state of an individual resource during runtime. The newly set protection state is persistent until the next call of the Xcp_Init function where all flags are set again.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > Seed&Key has to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpSeedKey	
Expected Caller Context	
> Task level	

Table 4-16 Xcp_ModifyProtectionStatus

4.3.15 Xcp_GetSessionStatus

Prototype	
`uint16 Xcp_GetSessionStatus ( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.

Return code	
uint16	The function returns a bit mask with the following flags: XCP_SESSION_CONNECTED: The XCP is in state connected. XCP_SESSION_POLLING: A polling measurement is ongoing. XCP_SESSION_DAQ: A DAQ measurement is active.
Functional Description	
This service can be used to get the session state of the XCP Protocol Layer. The session state is returned as a bit mask where the individual bits can be tested.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > Session Status API has to be enabled: /MICROSAR/Xcp/XcpGeneral/XcpSessionStatusAPI	
Expected Caller Context	
> Task level	

Table 4-17 Xcp_GetSessionStatus

4.3.16 Xcp_GetXcpDataPointer

Prototype	
`uint16 Xcp_GetXcpDataPointer ( Xcp_ChannelStructPtr * pXcpData )`	
Parameter	
pXcpData	Pointer to XCP channel information.
Return code	
–	-
Functional Description	
This service can be used to get the complete XCP data. This is required for flash programming with a flash kernel.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > Bootloader Download has to be enabled: /MICROSAR/Xcp/XcpGeneral/XcpBootloaderDownload	
Expected Caller Context	
> Task level	

Table 4-18 Xcp_GetXcpDataPointer

4.3.17 Xcp_SetStimMode

Prototype
`void Xcp_SetStimMode ( uint8 mode )`

Parameter	
`Mode`	The STIM mode to select. This can either be `XCP_STIM_SINGLE_SHOT_MODE:` Valid STIM data is written a single time (default). `XCP_STIM_CONTINUOUS_MODE:` Valid STIM data is written continuously.
Return code	
–	-
Functional Description	
This service is used to change the behavior of the Xcp_Event function when new STIM data is written.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > Data acquisition and STIM has to be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpSynchronousDataStim	
Expected Caller Context	
> Task level	

Table 4-19 Xcp_SetStimMode

4.3.18 Xcp_DaqSetComplementaryChannel

Prototype	
void **Xcp_DaqSetComplementaryChannel** (Xcp_ChannelType XcpChannel, boolean ChannelEnabled)	
Parameter	
XcpChannel	The channel number in multi client mode.
ChannelEnabled	Select TRUE when the complementary channel shall be enabled; FALSE for regular measurement.
Return code	
–	-
Functional Description	
This service is used to change the behavior of the Xcp_Event function to optionally transmit the data via the complementary channel. The complementary channel is a memory mapped interface used for high performance; high bandwidth data transfer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > Complementary channel must be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpComplementaryDAQ	
Expected Caller Context	
> Task level	

Table 4-20 Xcp_DaqSetComplementaryChannel

4.3.19 Xcp_EventGetDaqSize

Prototype	
Std_ReturnType **Xcp_EventGetDaqSize** (Xcp_ChannelType XcpChannel, uint16 EventChannel, uint32 *DaqSize, uint8 *DaqPriority)	
Parameter	
XcpChannel	The channel number in multi client mode.
EventChannel	Of this event channel number the size of the associated DAQ lists are calculated.
DaqSize	The returned size of all associated DAQ lists. This value can be used to allocate the memory mapped buffer.
DaqPriority	The returned priority of the DAQ lists.
Return code	
E_OK	The size was calculated successfully.
E_NOT_OK	The calculation failed, the DAQ list is not valid.
Functional Description	
This service is used to get the size of the associated DAQ list(s). The size can be used to allocate a memory buffer for transmission via the complementary channel.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > Complementary channel must be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpComplementaryDAQ
Expected Caller Context
> Task level

Table 4-21 Xcp_EventGetDaqSize

4.4 Services provided by the XCP Protocol Layer and called by the XCP Transport Layer

4.4.1 Xcp_TlMainFunction

Prototype	
`void Xcp_TlMainFunction ( uint8 ActiveTl )`	
Parameter	
`ActiveTl`	This parameter defines the channel for which transmission of DAQ messages shall be triggered.
Return code	
–	-
Functional Description	
This function is usually called by the transport layer to trigger transmission of DAQ frames. It is called channel wise. This function can also be used by the user to trigger transmission on request.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has been initialized correctly	
Expected Caller Context	
> Task level	

Table 4-22 Xcp_TlMainFunction

4.4.2 Xcp_TlRxIndication

Prototype	
`void` **`Xcp_TlRxIndication`** `( Xcp_ChannelType XcpChannel, unt8 *CmdPtr )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`CmdPtr`	Pointer to the XCP protocol message, which must be extracted from the XCP protocol packet.
Return code	
–	-
Functional Description	
Every time the XCP Transport Layer receives a XCP CTO Packet this function has to be called. The parameter is a pointer to the XCP protocol message, which must be extracted from the XCP protocol packet.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task level	

Table 4-23 Xcp_TlRxIndication

4.4.3 Xcp_TlTxConfirmation

Prototype	
`void` **`Xcp_TlTxConfirmation`** `( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
–	-
Functional Description	
The XCP Protocol Layer does not call `<Bus>Xcp_Send` again, until `Xcp_TlTxConfirmation` has confirmed the successful transmission of the previous message. `Xcp_TlTxConfirmation` transmits pending data acquisition messages by calling `<Bus>Xcp_Send` again. Note that if `Xcp_TlTxConfirmation` is called from inside `<Bus>Xcp_Send` a recursion occurs, which assumes enough space on the call stack.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	

Expected Caller Context
> Task level

Table 4-24 Xcp_TlTxConfirmation

4.4.4 Xcp_SetActiveTl

Prototype	
`void Xcp_SetActiveTl ( Xcp_ChannelType XcpChannel, uint8 MaxCto, uint16 MaxDto, uint8 ActiveTl )`	
Parameter	
`XcpChannel`	The channel number in multi-client mode.
`MaxCto`	Max CTO used by the respective XCP Transport Layer
`MaxDto`	Max DTO used by the respective XCP Transport Layer
`ActiveTl`	`XCP_TRANSPORT_LAYER_CAN`: XCP on CAN Transport Layer `XCP_TRANSPORT_LAYER_FR`: XCP on Fr Transport Layer `XCP_TRANSPORT_LAYER_ETH`: XCP on Ethernet Transport Layer
Return code	
–	-
Functional Description	
This service is used by the XCP Transport Layers to set the Transport Layer to be used by the XCP Protocol Layer	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task level	

Table 4-25 Xcp_SetActiveTl

4.4.5 Xcp_GetActiveTl

Prototype	
`uint8 Xcp_GetActiveTl ( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi-client mode.
Return code	
`uint8`	`XCP_TRANSPORT_LAYER_CAN`: XCP on CAN Transport Layer `XCP_TRANSPORT_LAYER_FR`: XCP on Fr Transport Layer `XCP_TRANSPORT_LAYER_ETH`: XCP on Ethernet Transport Layer

Functional Description
This service is used by the XCP Transport Layers to get the currently active Transport Layer used by the XCP Protocol Layer
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.
Expected Caller Context
> Task level

Table 4-26 Xcp_GetActiveTl

4.4.6 Xcp_TlQueryMessagePending

Prototype	
`boolean` **`Xcp_TlQueryMessagePending`** `( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi-client mode.
Return code	
`boolean`	TRUE: There is still another message in the queue. FALSE: Last message is pending.
Functional Description	
This service is used by the XCP Transport Layers to get the currently active message status.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task level	

Table 4-27 Xcp_TlQueryMessagePending

4.5 XCP Transport Layer Services called by the XCP Protocol Layer

4.5.1 <Bus>Xcp_Send

Prototype
`void <Bus>Xcp_Send ( Xcp_ChannelType XcpChannel, uint8 len, uint8 *msg )`

Parameter	
`XcpChannel`	The channel number in multi-client mode.
`len`	Length of message data
`msg`	Pointer to message
Return code	
-	-
Functional Description	
Requests for the transmission of a command transfer object (CTO) or data transfer object (DTO). `Xcp_TlTxConfirmation` must be called after the successful transmission of any XCP message. The XCP Protocol Layer will not request further transmissions otherwise.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task level	

Table 4-28 <Bus>Xcp_Send

4.5.2 <Bus>Xcp_SendFlush

Prototype	
`void <Bus>Xcp_SendFlush( Xcp_ChannelType XcpChannel, uint8 FlushType )`	
Parameter	
`XcpChannel`	The channel number in multi-client mode.
`FlushType`	This is one of the following: `XCP_FLUSH_CTO`: To flush CTO message. `XCP_FLUSH_DTO`: To flush DTO message. `XCP_FLUSH_ALL`: To flush either message.
Return code	
-	-
Functional Description	
Flush the transmit buffer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	

Expected Caller Context
> Task level

Table 4-29 <Bus>Xcp_SendFlush

4.5.3 <Bus>Xcp_TlService

Prototype	
`uint8 <Bus>Xcp_TlService( Xcp_ChannelType XcpChannel, uint8 *pCmd )`	
Parameter	
`XcpChannel`	The channel number in multi-client mode.
`pCmd`	Pointer to transport layer command string
Return code	
`uint8`	`XCP_CMD_OK` :Done `XCP_CMD_PENDING` : Call Xcp_SendCrm() when done `XCP_CMD_SYNTAX` : Error `XCP_CMD_BUSY` : not executed `XCP_CMD_UNKNOWN` : not implemented optional command `XCP_CMD_OUT_OF_RANGE` : command parameters out of range
Functional Description	
Transport Layer specific commands are processed within the XCP Transport Layer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task level	

Table 4-30 <Bus>Xcp_TlService

4.6 Application Services called by the XCP Protocol Layer

The prototypes of the functions that are required by the XCP Protocol Layer can be found in the XcpAppl header.

The XCP Protocol Layer provides application callback functions in order to perform application and hardware specific tasks.

Note: All services within this chapter are called from task or interrupt level. All services are not reentrant.

4.6.1 XcpAppl_GetTimestamp

Prototype	
`Xcp_TimestampType XcpAppl_GetTimestamp( void )`	
Parameter	
–	-
Return code	
`Xcp_TimestampType`	The timestamp which is either uint8, uint16 or uint32, depending on configuration.
Functional Description	
Returns the current timestamp.	
Particularities and Limitations	

> This function is synchronous.
> This function is non-reentrant.
> The XCP Protocol Layer has to be initialized correctly.
> DAQ and timestamp feature needs to be enabled:
`/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim`
`/MICROSAR/Xcp/XcpGeneral/XcpTimestampType`	

Expected Caller Context	
> Task level	

Table 4-31 XcpAppl_GetTimestamp

4.6.2 XcpAppl_GetPointer

Prototype	
`Xcp_AddressPtrType` **`XcpAppl_GetPointer`**`( Xcp_ChannelType XcpChannel, uint8 AddrExt, const Xcp_AddressPtrType Addr )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`AddrExt`	8 bit address extension
`Addr`	32 bit address
Return code	
`Xcp_AddressPtrType`	Pointer to the address specified by the parameters

Functional Description

This function converts a memory address from XCP format (32-bit address plus 8-bit address extension) to a C style pointer. An MCS like CANape usually reads this memory addresses from the ASAP2 database or from a linker map file.
The address extension may be used to distinguish different address spaces or memory types. In most cases, the address extension is not used and may be ignored.
This function is used to convert an address from the MCS tool.
Example:
The following code shows an example of a typical implementation of `XcpAppl_GetPointer`:
`Xcp_AddressPtrType XcpAppl_GetPointer( Xcp_ChannelType XcpChannel, uint8 AddrExt, uint32 Addr )`
`{`
`  return (Xcp_AddressPtrType)Addr;`
`}`

Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.
Expected Caller Context
> Task level

Table 4-32 XcpAppl_GetPointer

4.6.3 XcpAppl_GetIdData

Prototype	
`uint32 **XcpAppl_GetIdData**( uint8 **Data, uint8 Id )`	
Parameter	
`Data`	Pointer to location where address pointer to Id data is stored.
`Id`	Identification of the requested information/identification
Return code	
`uint32`	Length of the identification information.
Functional Description	
Returns a pointer to identification information as requested by the Xcp Master.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Get ID feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpGetIdGeneric`	
Expected Caller Context	
> Task level	

Table 4-33 XcpAppl_GetIdData

4.6.4 XcpAppl_GetSeedLength

Prototype	
`uint8 XcpAppl_GetSeedLength( const uint8 Resource )`	
Parameter	
`Resource`	Resource for which the seed length has to be generated `XCP_RM_CAL_PAG` : to unlock the resource calibration/paging `XCP_RM_DAQ` : to unlock the resource data acquisition `XCP_RM_STIM` : to unlock the resource stimulation `XCP_RM_PGM` : to unlock the resource programming
Return code	
`uint8`	The length of the generated seed.
Functional Description	
Return the length of the seed for the respective resource.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Seed&Key feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpSeedKey`	
Expected Caller Context	
> Task level	

Table 4-34 XcpAppl_GetSeedLength

4.6.5 XcpAppl_GetSeed

Prototype	
`void XcpAppl_GetSeed( const uint8 Resource, const uint8 Position, const uint8 Length, uint8 *Seed )`	
Parameter	
`Resource`	Resource for which the seed has to be generated `XCP_RM_CAL_PAG` : to unlock the resource calibration/paging `XCP_RM_DAQ` : to unlock the resource data acquisition `XCP_RM_STIM` : to unlock the resource stimulation `XCP_RM_PGM` : to unlock the resource programming
`Position`	The position of the seed fragment that needs to be copied.
`Length`	The length of the seed fragment that needs to be copied.
`Seed`	Pointer to RAM where the seed has to be copied to.
Return code	
–	

Functional Description
Generate a seed for the appropriate resource.
Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Seed&Key feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpSeedKey`
Expected Caller Context
> Task level

Table 4-35 XcpAppl_GetSeed

4.6.6 XcpAppl_UnlockWriteKey

Prototype	
`void XcpAppl_UnlockWriteKey( const uint8 *Key, const uint8 Position, const uint8 Length )`	
Parameter	
`Key`	Pointer to key.
`Position`	The position the key fragment shall be inserted to.
`Length`	Length of the key fragment.
Return code	
–	
Functional Description	
Assembles a complete key out of one or multiple fragments.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Seed&Key feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpSeedKey`	
Expected Caller Context	
> Task level	

Table 4-36 XcpAppl_UnlockWriteKey

4.6.7 XcpAppl_Unlock

Prototype	
`uint8 XcpAppl_Unlock( void )`	
Parameter	
–	
Return code	
`uint8`	`0` : if the key is not valid `XCP_RM_CAL_PAG` : to unlock the resource calibration/paging `XCP_RM_DAQ` : to unlock the resource data acquisition `XCP_RM_STIM` : to unlock the resource stimulation `XCP_RM_PGM` : to unlock the resource programming
Functional Description	
Check the key and return the resource that has to be unlocked. The complete key was previously assembled with XcpAppl_UnlockWriteKey.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Seed&Key feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpSeedKey`	
Expected Caller Context	
> Task level	

Table 4-37 XcpAppl_Unlock

4.6.8 XcpAppl_CalibrationWrite

Prototype	
`uint8 XcpAppl_CalibrationWrite( Xcp_AddressPtrType Dst, uint8 *Src, uint8 Size )`	
Parameter	
`Dst`	Destination address as integer.
`Src`	Pointer to source of data.
`Size`	Size of data to copy from Src to Dst.
Return code	
`uint8`	`XCP_CMD_DENIED` : if access is denied `XCP_CMD_PENDING` : access is performed asynchronously (e.g. EEPROM) `XCP_CMD_OK` : if access is granted
Functional Description	
Check addresses for valid write access and copy data from source to destination.	

Particularities and Limitations
> This function can be synchronous and asynchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.
Expected Caller Context
> Task level

Table 4-38 XcpAppl_CalibrationWrite

4.6.9 XcpAppl_MeasurementRead

Prototype	
`uint8 XcpAppl_MeasurementRead( uint8 *Dst, Xcp_AddressPtrType Src, uint8 Size )`	
Parameter	
`Dst`	Pointer to destination address
`Src`	Source address of data as integer
`Size`	Size of data to copy from Src to Dst.
Return code	
`uint8`	`XCP_CMD_DENIED` : if access is denied `XCP_CMD_PENDING` : access is performed asynchronously (e.g. EEPROM) `XCP_CMD_OK` : if access is granted
Functional Description	
Check addresses for valid read access and copy data from source to destination.	
Particularities and Limitations	
> This function can be synchronous and asynchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task level	

Table 4-39 XcpAppl_MeasurementRead

4.6.10 XcpAppl_CheckReadAccess

Prototype	
`uint8 XcpAppl_CheckReadAccess( Xcp_ChannelType XcpChannel, Xcp_AddressPtrType Address, uint32 Size )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`Address`	Destination address to check.
`Size`	Size of data to check.

Return code	
uint8	XCP_CMD_DENIED : if access is denied XCP_CMD_OK : if access is granted
Functional Description	
Check addresses for valid read access.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Read Protection feature need to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpMemoryReadProtection	
Expected Caller Context	
> Task level	

Table 4-40 XcpAppl_CheckReadAccess

4.6.11 XcpAppl_CheckProgramAccess

Prototype	
uint8 **XcpAppl_CheckProgramAccess**(Xcp_AddressPtrType Address, uint32 Size)	
Parameter	
Address	Destination address to check.
Size	Size of data to check.
Return code	
uint8	XCP_CMD_DENIED : if access is denied XCP_CMD_OK : if access is granted
Functional Description	
Check addresses for valid write flash access.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task level	

Table 4-41 XcpAppl_CheckProgramAccess

4.6.12 XcpAppl_UserService

Prototype
uint8 **XcpAppl_UserService**(uint8 *Cmd)

Parameter	
Cmd	Pointer to command string
Return code	
uint8	XCP_CMD_OK : if command is accepted. XCP_CMD_PENDING : if command is performed asynchronously. XCP_CMD_SYNTAX : if command is not accepted.
Functional Description	
Application specific user command.	
Particularities and Limitations	
> This function is asynchronous if it returns XCP_CMD_PENDING. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > User command feature need to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpUserDefinedCommand	
Expected Caller Context	
> Task level	

Table 4-42 XcpAppl_UserService

4.6.13 XcpAppl_OpenCmdIf

Prototype
uint8 **XcpAppl_OpenCmdIf**(Xcp_ChannelType XcpChannel, uint8 *Cmd, uint8 *Response, uint8 *Length)

Parameter	
XcpChannel	The channel number in multi client mode.
Cmd	Pointer to command string
Response	Pointer to response string
Length	Pointer to response length
Return code	
uint8	XCP_CMD_OK : if command is accepted. XCP_CMD_PENDING : if command is performed asynchronously. XCP_CMD_UNKNOWN : if command is not accepted.
Functional Description	
Call back that can be used to extend the XCP commands of the XCP protocol layer.	
Particularities and Limitations	
> This function is asynchronous if it returns XCP_CMD_PENDING. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > User command feature need to be enabled: /MICROSAR/Xcp/XcpCmdConfig/XcpOpenCommandInterface	

Expected Caller Context
> Task level

Table 4-43 XcpAppl_OpenCmdIf

4.6.14 XcpAppl_SendStall

Prototype	
`uint8 XcpAppl_SendStall( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
`uint8`	`0` : Reject sending of new message. `1` : continue processing.
Functional Description	
Resolve a transmit stall condition in `Xcp_Putchar` or `Xcp_SendEvent`.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Service request Messages feature need to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpAsynchMessage/XcpServiceRequestMessage`	
Expected Caller Context	
> Task level	

Table 4-44 XcpAppl_SendStall

4.6.15 XcpAppl_DisableNormalOperation

Prototype	
`uint8 XcpAppl_DisableNormalOperation( Xcp_AddressPtrType Address, uint16 Size )`	
Parameter	
`Address`	Address (where the flash kernel is downloaded to)
`Size`	Size (of the flash kernel)
Return code	
`uint8`	`XCP_CMD_OK:` download of flash kernel confirmed `XCP_CMD_DENIED:` download of flash kernel refused
Functional Description	
Prior to the flash kernel download has the ECU's normal operation to be stopped in order to avoid misbehavior due to data inconsistencies.	

Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Bootloader download feature need to be enabled: `/MICROSAR/Xcp/XcpGeneral/XcpBootloaderDownload`
Expected Caller Context
> Task level

Table 4-45 XcpAppl_DisableNormalOperation

4.6.16 XcpAppl_StartBootLoader

Prototype	
`uint8 XcpAppl_StartBootLoader( void )`	
Parameter	
–	-
Return code	
`uint8`	This function should not return. `XCP_CMD_OK` : positive response `XCP_CMD_BUSY` : negative response
Functional Description	
Start of the boot loader.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Bootloader download feature need to be enabled: `/MICROSAR/Xcp/XcpGeneral/XcpBootloaderDownload`	
Expected Caller Context	
> Task level	

Table 4-46 XcpAppl_StartBootLoader

4.6.17 XcpAppl_Reset

Prototype	
`void XcpAppl_Reset( void )`	
Parameter	
–	-

Return code	
–	-

Functional Description

Perform an ECU reset after reprogramming of the application.

Particularities and Limitations

> This function is synchronous.
> This function is non-reentrant.
> The XCP Protocol Layer has to be initialized correctly.
> Programming feature needs to be enabled:
`/MICROSAR/Xcp/XcpCmdConfig/XcpProgramming`

Expected Caller Context

> Task level

Table 4-47 XcpAppl_Reset

4.6.18 XcpAppl_ProgramStart

Prototype

```
uint8 XcpAppl_ProgramStart( void )
```

Parameter	
–	-

Return code	
uint8	XCP_CMD_OK : Preparation done XCP_CMD_PENDING : Call Xcp_SendCrm() when done XCP_CMD_ERROR : Flash programming not possible

Functional Description

Prepare the ECU for flash programming.

Particularities and Limitations

> This function is asynchronous if it returns XCP_CMD_PENDING.
> This function is non-reentrant.
> The XCP Protocol Layer has to be initialized correctly.
> Programming feature needs to be enabled:
`/MICROSAR/Xcp/XcpCmdConfig/XcpProgramming`

Expected Caller Context

> Task level

Table 4-48 XcpAppl_ProgramStart

4.6.19 XcpAppl_FlashClear

Prototype

`uint8 XcpAppl_FlashClear( uint8 *Address, uint32 Size )`	
Parameter	
`Address`	Address of memory area to clear
`Size`	Size of memory area to clear
Return code	
`uint8`	`XCP_CMD_OK` : Flash memory erase done `XCP_CMD_PENDING` : Call Xcp_SendCrm() when done `XCP_CMD_ERROR` : Flash memory erase error
Functional Description	
Clear the flash memory, before the flash memory will be reprogrammed.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Programming feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpProgramming`	
Expected Caller Context	
> Task level	

Table 4-49 XcpAppl_FlashClear

4.6.20 XcpAppl_FlashProgram

Prototype	
`uint8 XcpAppl_FlashProgram( const uint8 *Data, uint8 *Address, uint8 Size )`	
Parameter	
`Data`	Pointer to data.
`Address`	Address of memory to store data at.
`Size`	Size of data.
Return code	
`uint8`	`XCP_CMD_OK` : Flash memory programming finished `XCP_CMD_PENDING` : Flash memory programming in progress. `Xcp_SendCrm` has to be called when done.
Functional Description	
Program the cleared flash memory.	

Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Programming feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpProgramming`
Expected Caller Context
> Task level

Table 4-50 XcpAppl_FlashProgram

4.6.21 XcpAppl_DaqResume

Prototype	
`uint8 XcpAppl_DaqResume( Xcp_ChannelType XcpChannel, Xcp_ChannelStruct *Channel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`Channel`	Pointer to dynamic DAQ list structure
Return code	
`uint8`	Boolean flag whether valid DAQ list was restored.
Functional Description	
Resume the automatic data transfer. The whole dynamic DAQ list structure that had been stored in non-volatile memory within the service `XcpAppl_DaqResumeStore(..)` has to be restored to RAM.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Resume Mode feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpResumeMode`	
Expected Caller Context	
> Task level	

Table 4-51 XcpAppl_DaqResume

4.6.22 XcpAppl_DaqResumeStore

Prototype	
`void XcpAppl_DaqResumeStore( Xcp_ChannelType XcpChannel, const Xcp_ChannelStruct *Channel, uint8 MeasurementStart )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.

`Channel`	Pointer to dynamic DAQ list structure
`MeasurementStart`	If > 0 then set flag to start measurement during next init
Return code	
–	-
Functional Description	
This application callback service has to store the whole dynamic DAQ list structure in non-volatile memory for the DAQ resume mode. Any old DAQ list configuration that might have been stored in non-volatile memory before this command, must not be applicable anymore. After a cold start or reset the dynamic DAQ list structure has to be restored by the application callback service `XcpAppl_DaqResume(..)` when the flag MeasurementStart is > 0.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Resume Mode feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpResumeMode`	
Expected Caller Context	
> Task level	

Table 4-52 XcpAppl_DaqResumeStore

4.6.23 XcpAppl_DaqResumeClear

Prototype	
`void` **`XcpAppl_DaqResumeClear`**`( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
–	-
Functional Description	
The whole dynamic DAQ list structure that had been stored in non-volatile memory within the service `XcpAppl_DaqResumeStore(..)` has to be cleared.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Resume Mode feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpResumeMode`	
Expected Caller Context	
> Task level	

Table 4-53 XcpAppl_DaqResumeClear

4.6.24 XcpAppl_CalResumeStore

Prototype	
`boolean XcpAppl_CalResumeStore( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
`boolean`	If true the calibration page was stored.
Functional Description	
This application callback service has to store the current calibration data in non-volatile memory for the resume mode. After a cold start or reset the calibration data has to be restored by the application.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Resume Mode feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching/XcpFreezeMode`	
Expected Caller Context	
> Task level	

Table 4-54 XcpAppl_CalResumeStore

4.6.25 XcpAppl_GetCalPage

Prototype	
`uint8 XcpAppl_GetCalPage( uint8 Segment, uint8 Mode )`	
Parameter	
`Segment`	Logical data segment number
`Mode`	Access mode The access mode can be one of the following values: 1 : ECU access 2 : XCP access
Return code	
`uint8`	Logical data page number
Functional Description	
This function returns the logical number of the calibration data page that is currently activated for the specified access mode and data segment.	

Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Page switching feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching`
Expected Caller Context
> Task level

Table 4-55 XcpAppl_GetCalPage

4.6.26 XcpAppl_SetCalPage

Prototype	
`uint8 XcpAppl_SetCalPage( uint8 Segment, uint8 Page, uint8 Mode )`	
Parameter	
`Segment`	Logical data segment number
`Page`	Logical data page number
`Mode`	Access mode The access mode can be one of the following values: 1 : ECU access the given page will be used by the slave device application 2 : XCP access the slave device XCP driver will access the given page Both flags may be set simultaneously or separately.
Return code	
`uint8`	`XCP_CMD_OK` : Operation completed successfully `XCP_CMD_PENDING` : Call Xcp_SendCrm() when done `XCP_CRC_OUT_OF_RANGE` : segment out of range (only one segment supported) `XCP_CRC_PAGE_NOT_VALID` : Selected page not available `XCP_CRC_PAGE_MODE_NOT_VALID` : Selected page mode not available
Functional Description	
Switch pages, e.g. from reference page to working page.	
Particularities and Limitations	
> This function is asynchronous if it returns `XCP_CMD_PENDING`. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Page switching feature needs to be enabled: `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching`	
Expected Caller Context	
> Task level	

Table 4-56 XcpAppl_SetCalPage

4.6.27 XcpAppl_CopyCalPage

Prototype	
`uint8 XcpAppl_CopyCalPage( uint8 SrcSeg, uint8 SrcPage, uint8 DestSeg, uint8 DestPage )`	
Parameter	
`SrcSeg`	Source segment.
`SrcPage`	Source page.
`DestSeg`	Destination segment.
`DestPage`	Destination page.
Return code	
`uint8`	`XCP_CMD_OK` : Operation completed successfully `XCP_CMD_PENDING` : Call XcpSendCrm() when done `XCP_CRC_PAGE_NOT_VALID` : Page not available `XCP_CRC_SEGMENT_NOT_VALID` : Segment not available `XCP_CRC_WRITE_PROTECTED` : Destination page is write protected.
Functional Description	
Copying of calibration data pages. The pages are copied from source to destination.	
Particularities and Limitations	
> This function is asynchronous if it returns `XCP_CMD_PENDING`. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Page switching feature needs to be enabled: > `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching/XcpCopyPage`	
Expected Caller Context	
> Task level	

Table 4-57 XcpAppl_CopyCalPage

4.6.28 XcpAppl_SetFreezeMode

Prototype	
`void XcpAppl_SetFreezeMode( uint8 Segment, uint8 Mode )`	
Parameter	
`Segment`	Segment to set freeze mode
`Mode`	New freeze mode
Return code	
–	–
Functional Description	
Setting the freeze mode of a certain segment. Application must store the current freeze mode of each segment.	

Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Page switching and Freeze Mode feature needs to be enabled: > `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching/XcpFreezeMode`
Expected Caller Context
> Task level

Table 4-58 XcpAppl_SetFreezeMode

4.6.29 XcpAppl_GetFreezeMode

Prototype	
`uint8 XcpAppl_GetFreezeMode( uint8 Segment )`	
Parameter	
`Segment`	Segment to read freeze mode
Return code	
`uint8`	Return the current freeze mode, set by XcpAppl_SetFreezeMode().
Functional Description	
Reading the freeze mode of a certain segment. Application must store the current freeze mode of each segment and report it by the return value of this function.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. > Page switching and Freeze Mode feature needs to be enabled: > `/MICROSAR/Xcp/XcpCmdConfig/XcpPageSwitching/XcpFreezeMode`	
Expected Caller Context	
> Task level	

Table 4-59 XcpAppl_GetFreezeMode

4.6.30 XcpAppl_CalculateChecksum

Prototype	
`uint8 XcpAppl_CalculateChecksum( uint8 *MemArea, uint8 *Result, uint32 Length )`	
Parameter	
`MemArea`	Address pointer to memory area
`Result`	Pointer to response string
`Length`	Length of mem area, used for checksum calculation

Return code	
uint8	XCP_CMD_OK : CRC calculation performed successfully XCP_CMD_PENDING : Pending response, triggered by call of Xcp_SendCrm XCP_CMD_DENIED : CRC calculation not possible

Functional Description

Normally the XCP uses internal checksum calculation functions. If the internal checksum calculation does not fit the user requirements this call-back can be used to calculate the checksum by the application.

Particularities and Limitations

- This function is asynchronous if it returns XCP_CMD_PENDING.
- This function is non-reentrant.
- The XCP Protocol Layer has to be initialized correctly.
- CRC feature needs to be enabled:
- /MICROSAR/Xcp/XcpCmdConfig/XcpStandard/XcpCRC/XcpCustomCRC

Expected Caller Context

- Task level

Table 4-60 XcpAppl_CalculateChecksum

4.6.31 XcpAppl_ConStateNotification

Prototype

```
uint8 XcpAppl_ConStateNotification( Xcp_ChannelType XcpChannel, uint8 ConnectionState )
```

Parameter	
XcpChannel	The channel number in multi client mode.
ConnectionState	The new connection state (XCP_CON_STATE_RESUME, XCP_CON_STATE_DISCONNECTED, XCP_CON_STATE_CONNECTED).

Return code	
-	-

Functional Description

Notifies the application that the connection state has changed and which the new state is.

Particularities and Limitations

- This function is synchronous.
- This function is non-reentrant.
- The XCP Protocol Layer has to be initialized correctly.

Expected Caller Context

- Task and interrupt level

Table 4-61 XcpAppl_ConStateNotification

4.6.32 XcpAppl_MemCpy

Prototype	
`uint8 XcpAppl_MemCpy( uint8 * Dst, const uint8 * Src, uint16 Size )`	
Parameter	
`Dst`	The destination where the data is copied to.
`Src`	The source where the data is copied from.
`Size`	The number of byte to be copied.
Return code	
-	-
Functional Description	
Copies data from source to destination.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly.	
Expected Caller Context	
> Task and interrupt level	

Table 4-62 XcpAppl_MemCpy

4.6.33 XcpAppl_DaqPrepare

Prototype	
`uint8 XcpAppl_DaqPrepare( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
`XCP_CMD_OK`	The preparations was processed successfully.
`XCP_CMD_ERROR`	The preparations failed, e.g. the Vx1000 interface is not available.
Functional Description	
Prepares the complementary channel by e.g. requesting buffers.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > Complementary channel must be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpComplementaryDAQ	
Expected Caller Context	
> Task and interrupt level	

Table 4-63 XcpAppl_DaqPrepare

4.6.34 XcpAppl_DaqCleanup

Prototype	
`void XcpAppl_DaqCleanup( Xcp_ChannelType XcpChannel )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
Return code	
–	-
Functional Description	
Cleanup if dynamic DAQ lists are freed. Can be used to e.g. release buffers for complementary DAQ channel.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > Complementary channel must be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpComplementaryDAQ	
Expected Caller Context	
> Task and interrupt level	

Table 4-64 XcpAppl_DaqCleanup

4.6.35 XcpAppl_GetSendQueueRef

Prototype	
`void XcpAppl_GetSendQueueRef( Xcp_ChannelType XcpChannel, uint16 EventIndex, uint8 *SendQueuePtr )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`EventIndex`	The index of the event channel
`SendQueuePtr`	A pointer to the send queue used for complementary channel transmission. If no buffer is available, return a NULL_PTR.
Return code	
–	-
Functional Description	
Used to get a pointer to the buffer used for transmission of DAQ messages.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > Complementary channel must be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpComplementaryDAQ	

Expected Caller Context
> Task and interrupt level

Table 4-65 XcpAppl_GetSendQueueRef

4.6.36 XcpAppl_TransmitSendQueueElement

Prototype	
`void XcpAppl_TransmitSendQueueElement( Xcp_ChannelType XcpChannel, uint16 EventIndex )`	
Parameter	
`XcpChannel`	The channel number in multi client mode.
`EventIndex`	The index of the event channel
Return code	
_	-
Functional Description	
Used to trigger transmission of the filled buffer previously requested by XcpAppl_GetSendQueueRef.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > Complementary channel must be enabled /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpComplementaryDAQ	
Expected Caller Context	
> Task and interrupt level	

Table 4-66 XcpAppl_TransmitSendQueueElement

4.7 Services used by XCP

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

Component	API
DET	Det_ReportError
OS	GetCoreID

Table 4-67 Services used by the XCP

5 How To

5.1 Configuring multiple channels

The XCP is multi channel capable. It supports two modes that need to be taken into consideration:

1. Single Client Mode

   Multiple channels can be configured but only one channel can be active at a time. The channel receiving the CONNECT command is activated.

2. Multi Client Mode

   Multiple channels can be configured and be active independently at the same time.

To configure one channel two PDUs (one for RX and one for TX) must be present:

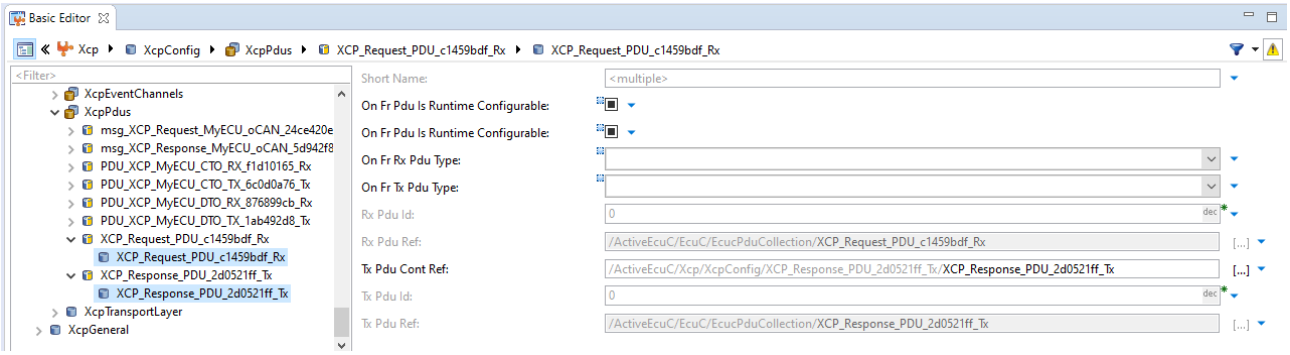


Figure 5-1 Configuring a Xcp channel.

The two PDUs are linked via /MICROSAR/Xcp/XcpConfig/XcpPdu/XcpRxPdu/XcpTxPduContRef. This reference defines the pair of PDUs used for one channel. Note that this mechanism is only relevant for CAN and Ethernet. On FlexRay there is only one channel and the PDU Type is selected.

5.2 Communication size parameters

The XCP provides several parameters that must be set according to the intended size of the measurement.

5.2.1 Max Number of Odt Daq

The parameter /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpMaxNumberOfOdtDaq defines the maximum number of individual ODT messages for DAQ.

As an example: If you have 10 Byte Variables to measure and are using standard CAN, each XCP message can transport a maximum of 7 measurement Variables. So, you would need a total of 2 ODT messages for the 10 Variables.

In general, it is safe to leave this parameter at its default (and max) value of 123 as the resource consumption of this parameter is low. This parameter plays a more important role for DAQ Message Header ODT_DAQ8 where much more ODTs are possible.

5.2.2 Max Number of ODT Entries

The parameter /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpMaxNumberOfOdtEntries defines the maximum number of measurement variables.

As an example: If you intend to measure 100 individual Variables of any size you would set this parameter to 100. This parameter defines a pool of measurement objects that are split between all DAQ lists.

5.2.3 Max Number of Odt Stim

The parameter /MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpMaxNumberOfOdtStim defines the maximum number of individual ODT message for STIM.

This parameter defines the number of ODTs as in 5.2.1 only for the other direction of STIM.

5.3 Resource Consumption

The ASAM 1.2 specification defines a2l extensions to support a resource consumption overview in the Tool. This feature is independent from the ECU implementation and completely modelled in the a2l. It is currently not generated by our a2l Generator but can be implemented manually.

5.3.1 ECU RAM Consumption

The RAM consumption is described in a table located in the IF_DATA XCP in the a2l (see chpt. 7.2.3 [5])

NAME	Description of Parameter
ODT_SIZE	0
ODT_ENTRY_SIZE	1
ODT_DAQ_BUFFER_ELEMENT_SIZE	0
ODT_STIM_BUFFER_ELEMENT_SIZE	0
DAQ_SIZE	0
DAQ_MEMORY_LIMIT	/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpMaxNumberOfOdtEntries

Table 5-1 ECU RAM Consumption

This table is limited to the Number of ODT Entries as the most important parameter. Parameters like ODT_SIZE or DAQ_SIZE can not be modelled due to the limitation of the a2l description.

5.3.2 CPU Execution Time

The CPU execution time is not supported by this TechRef. Determining the runtime behaviour of the XCP Slave is very platform/compiler/options dependent and difficult to obtain.

6 Configuration

6.1 Configuration Variants

The XCP supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the XCP parameters depend on the supported configuration variants. For their definitions please see the Xcp_bswmd.arxml file.

7 Glossary and Abbreviations

7.1 Abbreviations

Abbreviation	Description
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
BYP	**BYP**assing
CAN	**C**ontroller **A**rea **N**etwork
CAL	**CAL**ibration
CANape	Calibration and Measurement Data Acquisition for Electronic Control Systems
CMD	**C**o**m**man**d**
CTO	**C**ommand **T**ransfer **O**bject
DAQ	Synchronous **D**ata **Acq**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
ERR	**Err**or Packet
EV	**Ev**ent packet
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.
ODT	**O**bject **D**escriptor **T**able
OEM	**O**riginal **e**quipment **m**anufacturer (vehicle manufacturer)
PAG	**PAG**ing
PID	**P**acket **Id**entifier
PGM	**Pro**gra**m**ming
RAM	**R**andom **A**ccess **M**emory
RES	Command **Res**ponse Packet
ROM	**R**ead **O**nly **M**emory
SERV	**Ser**vice Request Packet
STIM	**Stim**ulation
TCP/IP	**T**ransfer **C**ontrol **P**rotocol / **I**nternet **P**rotocol

UDP/IP	**U**nified **D**ata **P**rotocol / **I**nternet **P**rotocol
USB	**U**niversal **S**erial **B**us
XCP	Universal Measurement and **C**alibration **P**rotocol
VI	**V**ector **I**nformatik GmbH

Table 7-1 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com