

Technical Reference

Version 2.0.0

Authors	Andreas Herkommer
Status	Released

Andreas Herkommer	2017-02-13	1.00.00	Initial Version
Andreas Herkommer	2017-11-14	2.00.00	Added new API Xcp_SetStimMode

[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_XCP_Part2-Protocol-Layer-Specification_V1-1-0.pdf	V1.1

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.



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.



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.

.....	
2.1	Architecture Overview 11
.....	
3.1	Features 13
3.1.1	Deviations 13
3.1.2	Additions/ Extensions 15
3.2	Initialization 15
3.3	States 15
3.4	Main Functions 16
3.5	Block Transfer Communication Model 16
3.6	Slave Device Identification 17
3.6.1	XCP Station Identifier 17
3.6.2	XCP Generic Identification 17
3.7	Seed & Key 17
3.8	Checksum Calculation 18
3.8.1	Custom CRC calculation 18
3.9	Memory Access by Application 18
3.9.1	Memory Read and Write Protection 18
3.9.2	Special use case “Type Safe Copy” 19
3.10	Event Codes 19
3.11	Service Request Messages 20
3.12	User Defined Command 20
3.13	Synchronous Data Transfer 20
3.13.1	Synchronous Data Acquisition (DAQ) 20
3.13.2	DAQ Timestamp 21
3.13.3	Power-Up Data Transfer 21
3.13.4	Data Stimulation (STIM) 22
3.13.5	Bypassing 22
3.13.6	Data Acquisition Plug & Play Mechanisms 22
3.13.7	Event Channel Plug & Play Mechanism 23
3.13.8	Send Queue 23
3.13.9	Data consistency 23
3.14	The Online Data Calibration Model 24
3.14.1	Page Switching 24
3.14.2	Page Switching Plug & Play Mechanism 24

3.14.3	Calibration Data Page Copying	24
3.14.4	Freeze Mode Handling	24
3.15	Flash Programming.....	25
3.15.1	Flash Programming by the ECU's Application	25
3.15.2	Flash Programming Plug & Play Mechanism	25
3.15.3	Flash Programming with a Flash Kernel.....	26
3.16	Multi Core Support.....	26
3.16.1	Type Safe Copy	26
3.16.2	DAQ/STIM with Multi Core	27
3.17	En- / Disabling the XCP module.....	28
3.18	XCP measurement during the post event time	28
3.19	Error Handling.....	29
3.19.1	Development Error Reporting.....	29
3.19.2	Production Code Error Reporting	30
.....		
4.1	Scope of Delivery.....	31
4.1.1	Static Files	31
4.1.2	Templates – user modifiable.....	31
4.1.3	Dynamic Files	31
4.1.4	Generated a2l files.....	31
4.2	Critical Sections	32
4.2.1	XCP_EXCLUSIVE_AREA_0	32
4.2.2	XCP_EXCLUSIVE_AREA_1	32
4.2.3	XCP_EXCLUSIVE_AREA_2	32
4.3	Memory Mapping	33
.....		
5.1	Type Definitions	34
5.2	Services provided by XCP	35
5.2.1	Xcp_InitMemory	35
5.2.2	Xcp_Init.....	35
5.2.3	Xcp_Event	36
5.2.4	Xcp_StimEventStatus	36
5.2.5	Xcp_MainFunction	37
5.2.6	Xcp_SendEvent	38
5.2.7	Xcp_PutChar.....	38
5.2.8	Xcp_Print	39
5.2.9	Xcp_Disconnect	39
5.2.10	Xcp_SendCrm.....	40
5.2.11	Xcp_GetVersionInfo	40

5.2.12	Xcp_ModifyProtectionStatus	41
5.2.13	Xcp_GetSessionStatus	42
5.2.14	Xcp_GetXcpDataPointer	42
5.2.15	Xcp_SetStimMode	43
5.3	Services provided by the XCP Protocol Layer and called by the XCP Transport Layer.....	43
5.3.1	Xcp_TIRxIndication	43
5.3.2	Xcp_TITxConfirmation.....	44
5.3.3	Xcp_SetActiveTI.....	44
5.3.4	Xcp_GetActiveTI	45
5.4	XCP Transport Layer Services called by the XCP Protocol Layer	46
5.4.1	<Bus>Xcp_Send	46
5.4.2	<Bus>Xcp_SendFlush	47
5.4.3	<Bus>Xcp_TIService.....	47
5.5	Application Services called by the XCP Protocol Layer	48
5.5.1	XcpAppl_GetTimestamp	49
5.5.2	XcpAppl_GetPointer.....	49
5.5.3	XcpAppl_GetIdData	50
5.5.4	XcpAppl_GetSeed	51
5.5.5	XcpAppl_Unlock.....	51
5.5.6	XcpAppl_CalibrationWrite	52
5.5.7	XcpAppl_MeasurementRead	53
5.5.8	XcpAppl_CheckReadAccess.....	53
5.5.9	XcpAppl_CheckProgramAccess.....	54
5.5.10	XcpAppl_UserService	54
5.5.11	XcpAppl_OpenCmdIf	55
5.5.12	XcpAppl_SendStall	55
5.5.13	XcpAppl_DisableNormalOperation.....	56
5.5.14	XcpAppl_StartBootLoader.....	57
5.5.15	XcpAppl_Reset	57
5.5.16	XcpAppl_ProgramStart	58
5.5.17	XcpAppl_FlashClear	58
5.5.18	XcpAppl_FlashProgram	59
5.5.19	XcpAppl_DaqResume.....	59
5.5.20	XcpAppl_DaqResumeStore	60
5.5.21	XcpAppl_DaqResumeClear	61
5.5.22	XcpAppl_CalResumeStore.....	61
5.5.23	XcpAppl_GetCalPage	62
5.5.24	XcpAppl_SetCalPage.....	62
5.5.25	XcpAppl_CopyCalPage.....	63
5.5.26	XcpAppl_SetFreezeMode	64

5.5.27	XcpAppl_GetFreezeMode	65
5.5.28	XcpAppl_CalculateChecksum	65
5.5.29	XcpAppl_ConStateNotification	66
5.5.30	XcpAppl_MemCpy	66
5.6	Services used by XCP	67
.....		
6.1	Configuration Variants.....	68
.....		
7.1	Abbreviations	69
.....		

Figure 2-1	AUTOSAR 4.1 Architecture Overview	11
Figure 2-2	Interfaces to adjacent modules of the XCP	12
Figure 3-1	Connection State Machine	16
Figure 3-2	Data consistency	23
Figure 3-3	Application of Xcp_Event function on Multi Core systems	28

Table 1-1	Component history	10
Table 3-1	Supported AUTOSAR standard conform features	13
Table 3-2	Deviations from AUTOSAR standard conform features	13
Table 3-3	Deviations from ASAM standard conform features	15
Table 3-4	Features provided beyond the AUTOSAR standard	15
Table 3-5	States	15
Table 3-6	Event codes	20
Table 3-7	Service IDs	29
Table 3-8	Errors reported to DET	30
Table 3-9	Errors reported to DEM	30
Table 4-1	Static files	31
Table 4-2	Templates	31
Table 4-3	Generated files	31
Table 5-1	Type definitions	34
Table 5-2	Xcp_ChannelStruct	34
Table 5-3	Xcp_InitMemory	35
4	Xcp_Init	35
Table 5-5	Xcp_Event	36
Table 5-6	Xcp_StimEventStatus	37
Table 5-7	Xcp_MainFunction	37
Table 5-8	Xcp_SendEvent	38
Table 5-9	Xcp_PutChar	39
Table 5-10	Xcp_Print	39
Table 5-11	Xcp_Disconnect	40
Table 5-12	Xcp_SendCrm	40
Table 5-13	Xcp_GetVersionInfo	41
Table 5-14	Xcp_ModifyProtectionStatus	41
Table 5-15	Xcp_GetSessionStatus	42
Table 5-16	Xcp_GetXcpDataPointer	43
Table 5-17	Xcp_SetStimMode	43
Table 5-18	Xcp_TIRxIndication	44
Table 5-19	Xcp_TITxConfirmation	44
Table 5-20	Xcp_SetActiveTI	45
Table 5-21	Xcp_GetActiveTI	46
Table 5-22	<Bus>Xcp_Send	46
Table 5-23	<Bus>Xcp_SendFlush	47
Table 5-24	<Bus>Xcp_TIService	48
Table 5-25	XcpAppl_GetTimestamp	49
Table 5-26	XcpAppl_GetPointer	50
Table 5-27	XcpAppl_GetIdData	51
Table 5-28	XcpAppl_GetSeed	51
Table 5-29	XcpAppl_Unlock	52
Table 5-30	XcpAppl_CalibrationWrite	52

Table 5-31	XcpAppl_MeasurementRead	53
Table 5-32	XcpAppl_CheckReadAccess	54
Table 5-33	XcpAppl_CheckProgramAccess	54
Table 5-34	XcpAppl_UserService	55
Table 5-35	XcpAppl_OpenCmdIf	55
Table 5-36	XcpAppl_SendStall	56
Table 5-37	XcpAppl_DisableNormalOperation	56
Table 5-38	XcpAppl_StartBootLoader	57
Table 5-39	XcpAppl_Reset	58
Table 5-40	XcpAppl_ProgramStart	58
Table 5-41	XcpAppl_FlashClear	59
Table 5-42	XcpAppl_FlashProgram	59
Table 5-43	XcpAppl_DaqResume	60
Table 5-44	XcpAppl_DaqResumeStore	61
Table 5-45	XcpAppl_DaqResumeClear	61
Table 5-46	XcpAppl_CalResumeStore	62
Table 5-47	XcpAppl_GetCalPage	62
Table 5-48	XcpAppl_SetCalPage	63
Table 5-49	XcpAppl_CopyCalPage	64
Table 5-50	XcpAppl_SetFreezeMode	64
Table 5-51	XcpAppl_GetFreezeMode	65
Table 5-52	XcpAppl_CalculateChecksum	66
Table 5-53	XcpAppl_ConStateNotification	66
Table 5-54	XcpAppl_MemCpy	67
Table 5-55	Services used by the XCP	67
Table 7-1	Abbreviations	70

1

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

1.00.xx	Initial Version of re-factored AR4 Protocol Layer.
2.00.xx	Series production of MultiCore feature.
3.00.xx	Bugfixes and Continuous STIM feature.

Table 1-1 Component history

2

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

	4	
	pre-compile	
	XCP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
	XCP_MODULE_ID	212 decimal (according to ref. [4])

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

2.1 Architecture Overview

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

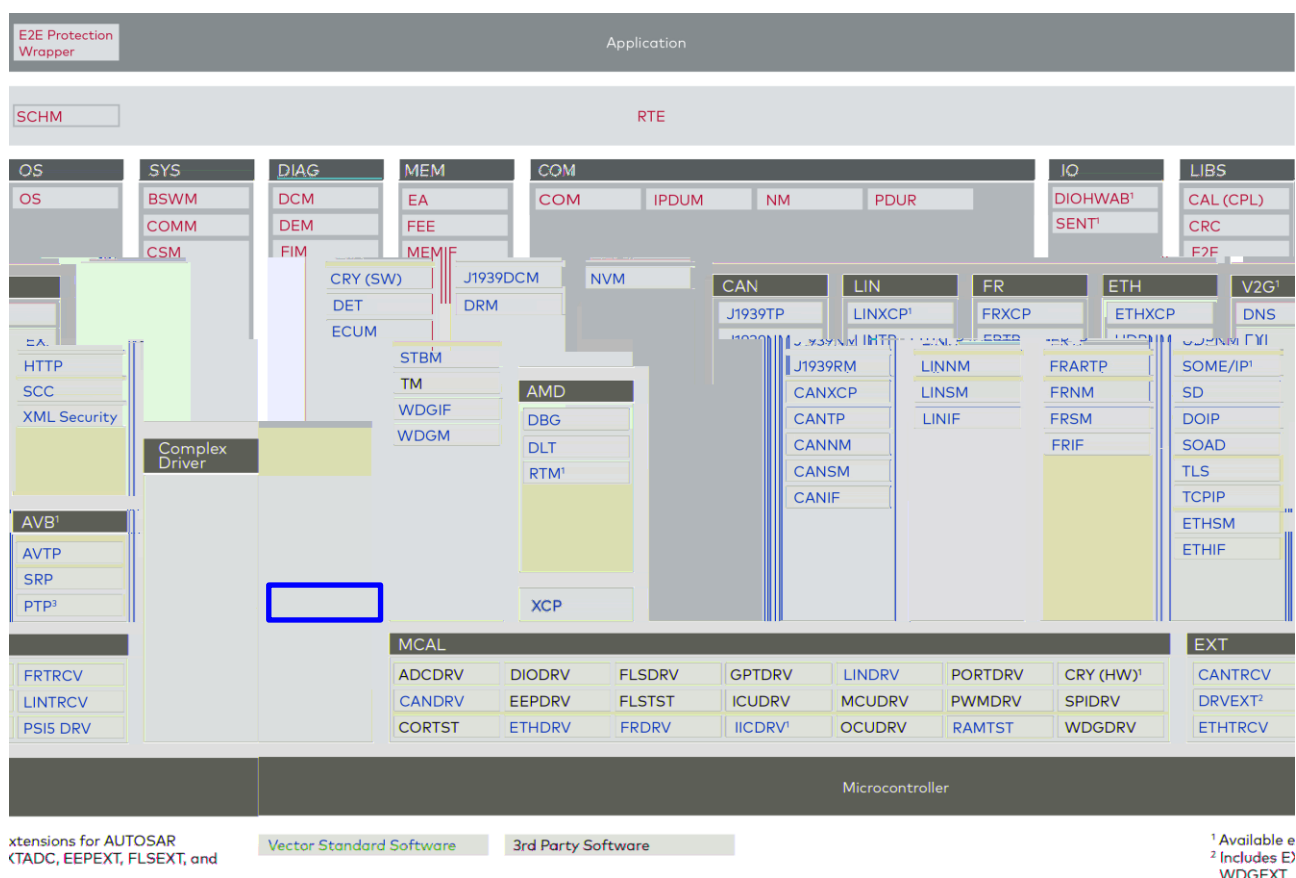


Figure 2-1 AUTOSAR 4.1 Architecture Overview

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 5.

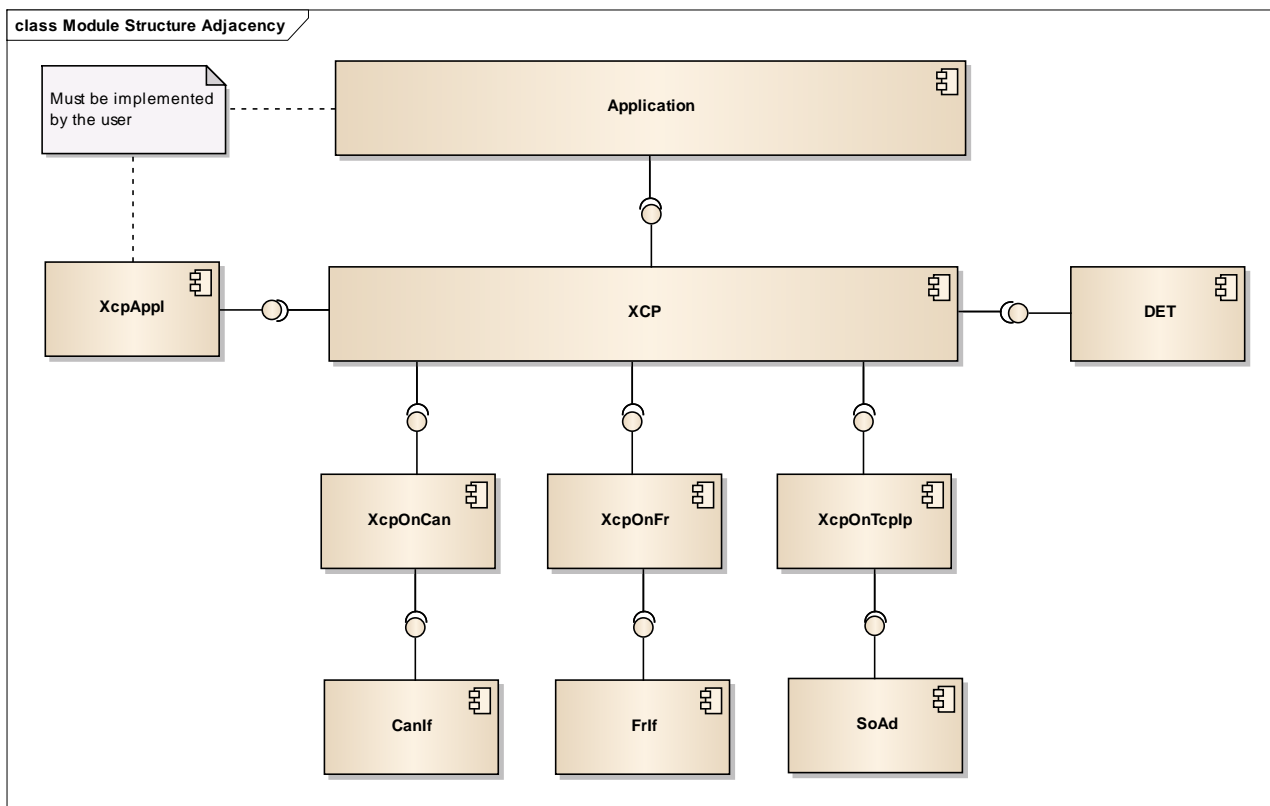


Figure 2-2 Interfaces to adjacent modules of the XCP

3

3.1 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 3-1 Supported AUTOSAR standard conform features
- > Table 3-2 Deviations from AUTOSAR standard conform features
- > Table 3-3 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 3-4 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

ASAM XCP Version 1.1		

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

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

Functional	The following features are not supported: <ul style="list-style-type: none">• The command <code>GET_SLAVE_ID</code>• A CDD as transport layer	4.2.2
API	The following APIs are not provided by XCP: <ul style="list-style-type: none">• <code>Xcp_SetTransmissionMode</code>	4.2.2
API	The API <code>Xcp_<Module>TriggerTransmit</code> is only supported for transport layer <code>FrIf</code> .	4.2.2

Table 3-2 Deviations from AUTOSAR standard conform features

Functional	1.6.4.1.2.4 Get general information on DAQ processor:	1.1

	<ul style="list-style-type: none"> Bitwise stimulation is not supported 	
Functional	1.6.4.2 Static DAQ list configuration (stat): <ul style="list-style-type: none"> Static DAQ lists are not supported; only dynamic DAQ lists are supported 	1.1
Functional	1.7.2.3 Interleaved Communication Model: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Only the default programming format is supported, therefore the command <code>PROGRAM_FORMAT</code> is not supported 	1.1
Functional	1.6.5.2.2 Get specific information for a sector: <ul style="list-style-type: none"> The command <code>GET_SECTOR_INFO</code> does not return a Program Sequence Number 	1.1
Functional	1.6.5.2.7 Program Verify: <ul style="list-style-type: none"> The command <code>PROGRAM_VERIFY</code> is not supported 	1.1
Functional	Daq configuration: <ul style="list-style-type: none"> Number of DAQ lists is limited to 0xFF Maximum DTO length is limited to 0xFF DAQ does not support address extension 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 <code>GET_DAQ_LIST_MODE</code>) 	1.1
Functional	5.1.10 ODT Optimization: <ul style="list-style-type: none"> The ODT Optimization is not supported 	1.2
Functional	1.2 Table of Event Codes: <ul style="list-style-type: none"> 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): <ul style="list-style-type: none"> The Service Request <code>SERV_RESET</code> is not supported 	1.1
Functional	1.6.1.2.9 Build Checksum over memory range: <ul style="list-style-type: none"> The checksum type <code>XCP_CRC_16</code> or <code>XCP_CRC_32</code> 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): <ul style="list-style-type: none"> The command <code>GET_PAGE_INFO</code> is not supported The command <code>GET_SEGMENT_INFO</code> is not supported Only one segment and two pages are supported 	1.1
Functional	Seed and Key: <ul style="list-style-type: none"> The seed size and key size must be equal or less <code>MAX_CTO-2</code> 	1.1

Functional	Consistency only supported on ODT level.	1.1
Functional	No other identification field type supported than “absolute ODT number”.	1.1

Table 3-3 Deviations from ASAM standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Support of CAN-FD
Support transmission and reception of DTO on multiple cores simultaneously.

Table 3-4 Features provided beyond the AUTOSAR standard

3.2 Initialization

The XCP gets initialized by call of the following services:

- 5.2.1 Xcp_InitMemory
- 5.2.2 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.

3.3 States

The XCP’s connection state machine is shown in Figure 3-1, comprises the following states:

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 3-5 States

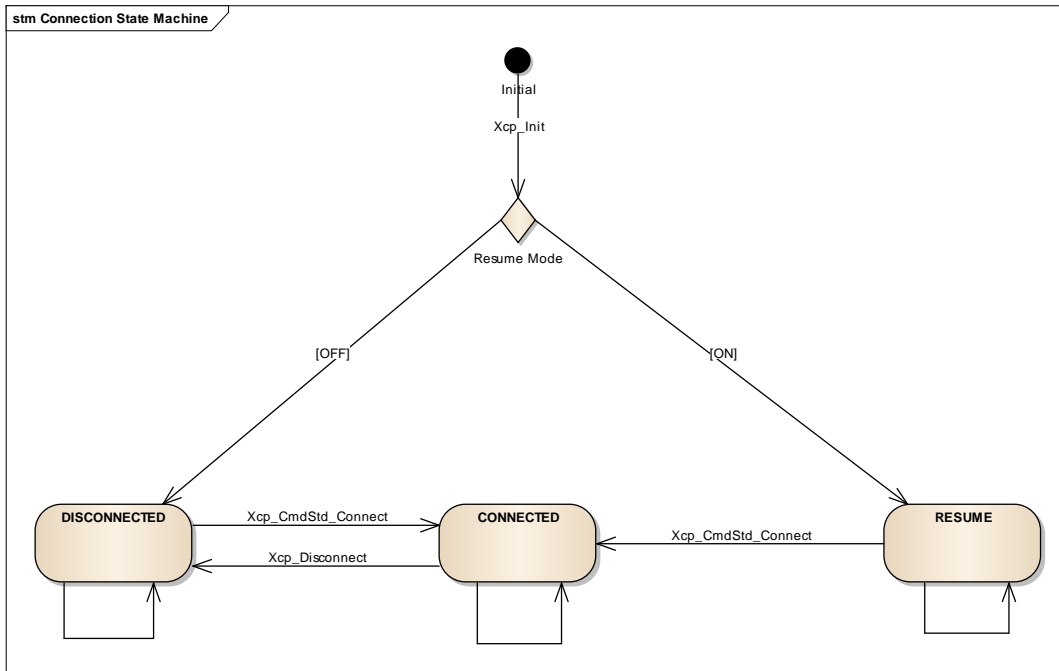


Figure 3-1 Connection State Machine

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:

- 5.2.9 Xcp_Disconnect

3.4 Main Functions

The Xcp provides a MainFunction:

- 5.2.5 Xcp_MainFunction

It must be called cyclically and performs the following tasks:

- > 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.

3.5 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.

3.6 Slave Device Identification

3.6.1 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 power-on start of the MCS via the plug & play mechanism. The plug & play mechanism prevents the user from selecting the wrong ECU database.

3.6.2 XCP Generic Identification

The XCP provides a generic mechanism for identification by the `GET_ID` command. For this purpose a callack exists.



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.

3.8 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.

3.8.1 Custom CRC calculation

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

- 5.5.28 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.

3.9 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:

- 5.5.6 XcpAppl_CalibrationWrite
- 5.5.7 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.

3.9.1 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:

- 5.5.8 XcpAppl_CheckReadAccess

This call-back is required for other services like CRC calculation to check the requested memory size beforehand.

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:

- 5.5.9 XcpAppl_CheckProgramAccess

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

3.9.2 S

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).

3.10 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:

- 5.2.6 Xcp_SendEvent.

The event codes can be found in the following table.

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 3-6 Event codes

3.11 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:

- 5.2.7 Xcp_PutChar
- 5.2.8 Xcp_Print

3.12 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:

- 5.5.10 XcpAppl_UserService

3.13 Synchronous Data Transfer

3.13.1 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(x)` has 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.

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.

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



Annotation for the usage of CANape:

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

3.13.2 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:

- 5.5.1 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 μ C, configured to a certain repetition rate (e.g. 1 μ s increment) for applications that require a high resolution of the timestamp and returned in the above mentioned call-back.

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

3.13.3 Power-Up Data Transfer

Power-up data transfer (also called **Power-Up Data Transfer (PUDT)** 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 have 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.

- 5.5.19 XcpAppl_DaqResume
- 5.5.20 XcpAppl_DaqResumeStore
- 5.5.21 XcpAppl_DaqResumeClear

To use the resume mode the compiler switch `/MICROSAR/Xcp/XcpCmdConfig/XcpDaqAndStim/XcpResumeMode` has 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`.



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.

3.13.4 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` has to be enabled.

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

3.13.5 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 has to be performed in a 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.

3.13.6 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

3.13.7 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.

3.13.8 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 3.16 Multi Core Support.

3.13.9 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()`

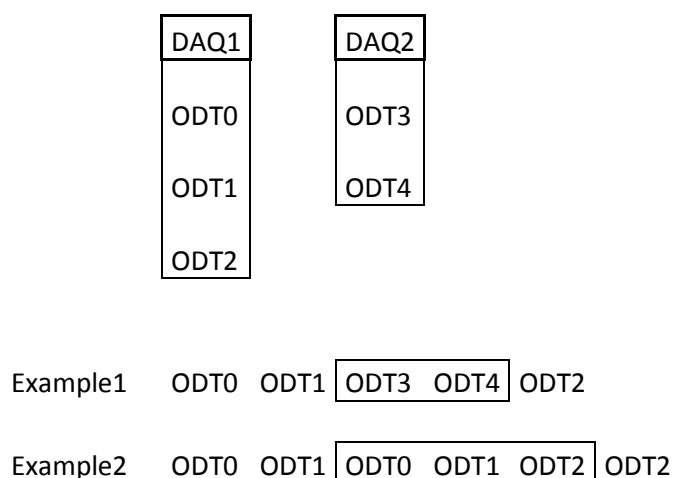


Figure 3-2 Data consistency

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.

3.14 The Online Data Calibration Model

3.14.1 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 have to be implemented:

- 5.5.23 XcpAppl_GetCalPage
- 5.5.24 XcpAppl_SetCalPage



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.

3.14.2 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`.

3.14.3 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 has to be provided by the application:

- 5.5.25 XcpAppl_CopyCalPage

3.14.4 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:

- 5.5.26 XcpAppl_SetFreezeMode

- 5.5.27 XcpAppl_GetFreezeMode
- 5.5.22 XcpAppl_CalResumeStore

3.15 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.

3.15.1 Flash Pro

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.

- 5.5.15 XcpAppl_Reset
- 5.5.16 XcpAppl_ProgramStart
- 5.5.17 XcpAppl_FlashClear
- 5.5.18 XcpAppl_FlashProgram

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



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.

3.15.2 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:

¹ Command not supported

- > 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`.

3.15.3 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

- 5.5.13 XcpAppl_DisableNormalOperation
- 5.5.14 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`.



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 *fk/* file for the microcontroller being used.

3.16 Multi Core Support

3.16.1 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.

3.16.2 DAQ/STIM with Multi Core

It is possible to execute the Xcp_Event 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:

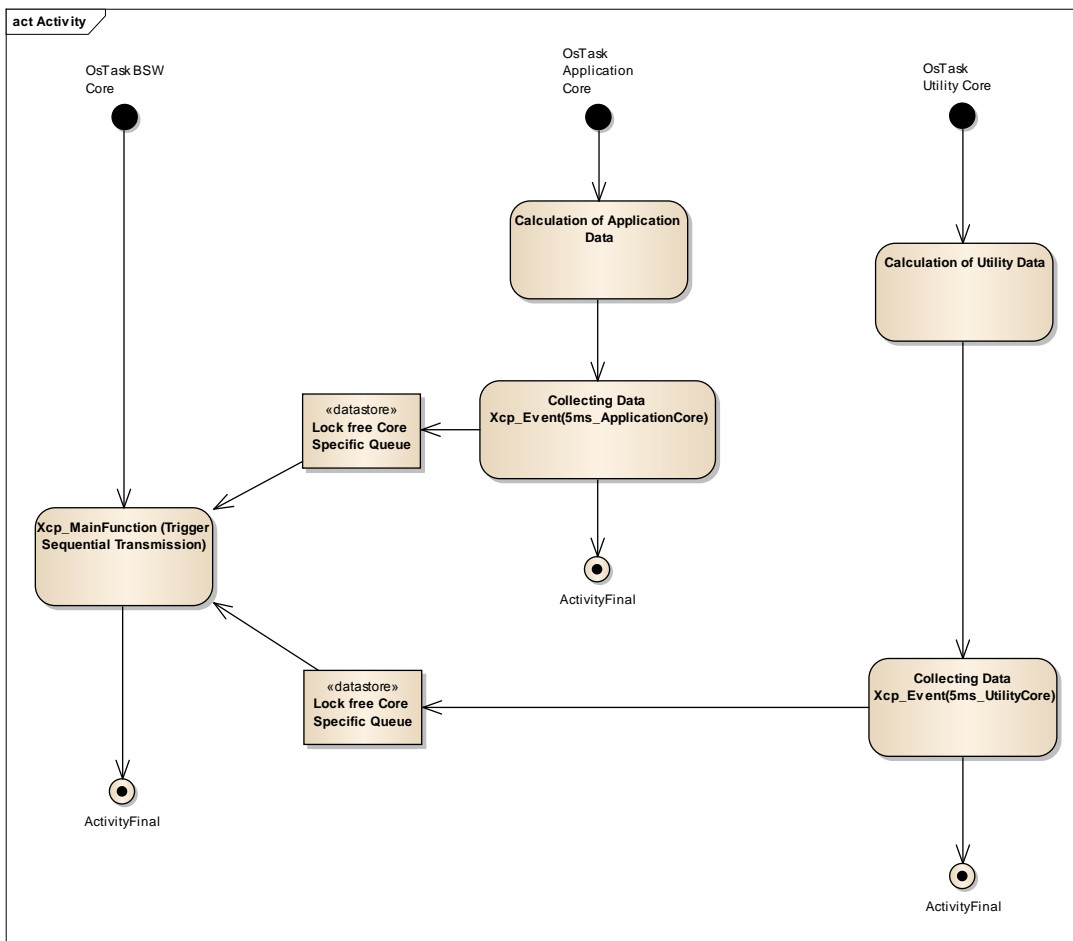


Figure 3-3 Application of Xcp_Event function on Multi Core systems

3.17 En- / Disabling the XCP module


The macro `XCP_ACTIVATE/XCP_DEACTIVATE` can be used to en- or disable the XCP module during run time. Thus the XCP functionality can be controlled by the application. These macros control the protocol and transport layer together, i.e. enabling or disabling 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 disabled.

3.18 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:

- 5.2.13 Xcp_GetSessionStatus

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

3.19 Error Handling

3.19.1 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 5.2. The following table presents the service IDs and the related services:

0x00	Xcp_Init
0x03	Xcp_SendEvent
0x04	Xcp_PutChar
0x05	Xcp_Print
0x06	Xcp_Disconnect
0x07	Xcp_SendCrm
0x08	Xcp_GetXcpDataPointer
0x0H	

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 3-8 Errors reported to DET

3.19.2 Production Code Error Reporting

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

-	No production errors are reported by the XCP.

Table 3-9 Errors reported to DEM

4

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

4.1 Scope of Delivery

The delivery of the XCP contains the files which are described in the chapters 4.1.1 and 4.1.3:

4.1.1 Static Files

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 4-1 Static files

4.1.2 Templates user modifiable

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 4-2 Templates

4.1.3 Dynamic Files

The dynamic files are generated by the configuration tool.

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 4-3 Generated files

4.1.4 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)

```
...  
/begin IF_DATA XCP  
  /include XCP.a21  
  /begin DAQ  
    /include XCP_daq.a21  
    /include XCP_events.a21  
    /include XCP_checks.a21  
  ...  
/end
```

4.2 Critical

The XCP protocol layer makes use of critical sections that are not re-entrant. The following sections

- 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:

4.2.1 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.

4.3 Memory Mapping

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



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

5

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

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

	-	
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.

Table 5-1 Type definitions

Xcp_ChannelStruct

	-	
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.

Table 5-2 Xcp_ChannelStruct

5.2 Services provided by XCP

5.2.1 Xcp_InitMemory

<code>void Xcp_InitMemory (void)</code>	
-	-
-	-
<p>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.</p>	
<ul style="list-style-type: none">> 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 5-3 Xcp_InitMemory

5.2.2 Xcp_Init

<code>void Xcp_Init (void)</code>	
-	-
-	-
<p>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).</p>	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is synchronous.> This function is non-reentrant.	
Expected Caller Context	
> Task level	

4 Xcp_Init

5.2.3 Xcp_Event

<code>uint8 Xcp_Event (uint16 EventChannel)</code>	
EventChannel	Number of event channels to process. The event channel numbers have to start at 0 and have to be continuous. The range is: 0..x
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
<p>Calling <code>Xcp_Event</code> with a particular event channel number triggers the sampling and transmission of all DAQ lists that are assigned to this event channel.</p> <p>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.</p> <p>Example:</p> <p>A motor control unit may have a 10ms, a 100ms and a crank synchronous event channel. In this case, the three <code>Xcp_Event</code> calls have to be placed at the appropriate locations in the ECU's program:</p> <pre>Xcp_Event (XcpConf_XcpEventChannel_10ms); /* 10ms cycle */ xcp_Event (XcpConf_XcpEventChannel_100ms); /* 100ms cycle */ xcp_Event (XcpConf_XcpEventChannel_Crank); /* Crank synchronous cycle */</pre>	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > Task and interrupt level 	

Table 5-5 Xcp_Event

5.2.4 Xcp_StimEventStatus

<code>uint8 Xcp_StimEventStatus (uint16 EventChannel, uint8 Action)</code>	
EventChannel	Event channel number.

Action	STIM_CHECK_ODT_BUFFER : check ODT buffer STIM_RESET_ODT_BUFFER : reset ODT buffer
uint8	: stimulation data not available : new stimulation data is available
Check if data stimulation (STIM) event can perform or delete the buffers.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task and interrupt level	

Table 5-6 Xcp_StimEventStatus

5.2.5 Xcp_MainFunction

<pre>void Xcp_MainFunction (void)</pre>	
-	-
-	-
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 <code>Xcp_MainFunction</code> . Additionally, the main function handles persisting requests.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-7 Xcp_MainFunction

5.2.6 Xcp_SendEvent

<pre>void Xcp_SendEvent (Xcp_ChannelType XcpChannel, uint8 EventCode, uint8 *EventData, uint8 Length)</pre>	
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.
-	-
Transmission of event codes via event packets (EV).	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-8 Xcp_SendEvent

5.2.7 Xcp_PutChar

<pre>void Xcp_PutChar (Xcp_ChannelType XcpChannel, uint8 *Character)</pre>	
XcpChannel	The channel number in multi client mode.
Character	The char to send.
-	-
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.	

- > 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 5-9 Xcp_PutChar

5.2.8 Xcp_Print

```
void Xcp_Print ( Xcp_ChannelType XcpChannel, uint8 *Str )
```

XcpChannel	The channel number in multi client mode.
Str	The 0 terminated string to send.

-	-
---	---

Transmission of a service request packet (SERV).
The string `str` is sent via service request packets. The string has to be terminated by 0x00.

- > 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 5-10 Xcp_Print

5.2.9 Xcp_Disconnect

```
void Xcp_Disconnect ( Xcp_ChannelType XcpChannel )
```

XcpChannel	The channel number in multi client mode.
------------	--

-	-
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.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-11 Xcp_Disconnect

5.2.10 Xcp_SendCrm

<code>void Xcp_SendCrm (Xcp_ChannelType XcpChannel)</code>	
XcpChannel	The channel number in multi client mode.
-	-
Transmission of a command response packet (RES), or error packet (ERR) if no other packet is pending.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-12 Xcp_SendCrm

5.2.11 Xcp_GetVersionInfo

<code>void Xcp_GetVersionInfo (Std_VersionInfoType *versionInfo)</code>	
versionInfo	Pointer to the location where the Version information shall be stored.

-	-
Xcp_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-13 Xcp_GetVersionInfo

5.2.12 Xcp_ModifyProtectionStatus

<pre>void Xcp_ModifyProtectionStatus (Xcp_ChannelType XcpChannel, uint8 AndState, uint8 OrState)</pre>	
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.
-	-
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.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-14 Xcp_ModifyProtectionStatus

5.2.13 Xcp_GetSessionStatus

uint16 Xcp_GetSessionStatus (Xcp_ChannelType XcpChannel)	
XcpChannel	The channel number in multi client mode.
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.
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.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-15 Xcp_GetSessionStatus

5.2.14 Xcp_GetXcpDataPointer

uint16 Xcp_GetXcpDataPointer (pXcpData)	
pXcpData	Pointer to XCP channel information.
-	-
This service can be used to get the complete XCP data. This is required for flash programming with a flash kernel.	
<ul style="list-style-type: none">> 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 5-16 Xcp_GetXcpDataPointer

5.2.15 Xcp_SetStimMode

```
void Xcp_SetStimMode ( uint8 mode )
```

Mode	The STIM mode to select. This can either be Valid STIM data is written a single time (default). Valid STIM data is written continuously.
------	---

-	-
---	---

This service is used to change the behavior of the Xcp_Event function when new STIM data is written.

- > 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 5-17 Xcp_SetStimMode

5.3 Services provided by the XCP Protocol Layer and called by the XCP Transport Layer**5.3.1 Xcp_TlRxIndication**

```
void Xcp_TlRxIndication ( Xcp_ChannelType XcpChannel, uint8 *CmdPtr )
```

XcpChannel	The channel number in multi client mode.
CmdPtr	Pointer to the XCP protocol message, which must be extracted from the XCP protocol packet.

-	-
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.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-18 Xcp_TIRxIndication

5.3.2 Xcp_TITxConfirmation

<pre>void Xcp_TITxConfirmation (Xcp_ChannelType XcpChannel)</pre>	
XcpChannel	The channel number in multi client mode.
-	-
The XCP Protocol Layer does not call <Bus>Xcp_Send again, until Xcp_TITxConfirmation has confirmed the successful transmission of the previous message. Xcp_TITxConfirmation transmits pending data acquisition messages by calling <Bus>Xcp_Send again. Note that if Xcp_TITxConfirmation is called from inside <Bus>Xcp_Send a recursion occurs, which assumes enough space on the call stack.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-19 Xcp_TITxConfirmation

5.3.3 Xcp_SetActiveTI

--	--

void Xcp_SetActiveTl (Xcp_ChannelType XcpChannel, uint8 MaxCto, uint16 MaxDto, uint8 ActiveTl)	
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
-	-
This service is used by the XCP Transport Layers to set the Transport Layer to be used by the XCP Protocol Layer	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > Task level 	

Table 5-20 Xcp_SetActiveTl

5.3.4 Xcp_GetActiveTl

uint8 Xcp_GetActiveTl (Xcp_ChannelType XcpChannel)	
XcpChannel	The channel number in multi client mode.
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
This service is used by the XCP Transport Layers to get the currently active Transport Layer used by the XCP Protocol Layer	

- > 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

5.4.2 <Bus>Xcp_SendFlush

void <Bus>Xcp_SendFlush(Xcp_ChannelType XcpChannel, uint8 FlushType)	
XcpChannel	The channel number in multi client mode.
FlushType	This is one of the following: XCP_FLUSH_CTO: To flush CTO messages. XCP_FLUSH_DTO: To flush DTO message. XCP_FLUSH_ALL: To flush either message.
-	-
Flush the transmit buffer.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > Task level 	

Table 5-23 <Bus>Xcp_SendFlush

5.4.3 <Bus>Xcp_TlService

uint8 <Bus>Xcp_TlService(Xcp_ChannelType XcpChannel, uint8 *pCmd)	
XcpChannel	The channel number in multi client mode.
pCmd	Pointer to transport layer command string
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
Transport Layer specific commands are processed within the XCP Transport Layer.	

<ul style="list-style-type: none"> > 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
<ul style="list-style-type: none"> > Task level

Table 5-24 <Bus>Xcp_TIService

5.5 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.

5.5.1 XcpAppl_GetTimestamp

<pre>Xcp_TimestampType XcpAppl_GetTimestamp(void)</pre>	
-	-
Xcp_TimestampType	The timestamp which is either uint8, uint16 or uint32, depending on configuration.
Returns the current timestamp.	
<ul style="list-style-type: none">> 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 5-25 XcpAppl_GetTimestamp

5.5.2 XcpAppl_GetPointer

<pre>Xcp_AddressPtrType XcpAppl_GetPointer(Xcp_ChannelType XcpChannel, uint8 AddrExt, const Xcp_AddressPtrType Addr)</pre>	
XcpChannel	The channel number in multi client mode.
AddrExt	8 bit address extension
Addr	32 bit address
Xcp_AddressPtrType	Pointer to the address specified by the parameters

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;
}
```

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

Expected Caller Context

> Task level

Table 5-27 XcpAppl_GetIdData

5.5.4 XcpAppl_GetSeed

```
uint8 XcpAppl_GetSeed( const uint8 Resource, uint8 *Seed )
```

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
----------	--

Seed	Pointer to RAM where the seed has to be generated to.
------	---

uint8	The length of the generated seed that is returned by seed.
-------	--

Generate a seed for the appropriate resource.
The seed has a maximum length of MAX_CTO-2 bytes.

- > 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 5-28 XcpAppl_GetSeed

5.5.5 XcpAppl_Unlock

```
uint8 XcpAppl_Unlock( const uint8 *Key, const uint8 Length )
```

Key	Pointer to key.
-----	-----------------

Length	Length of the key.
--------	--------------------

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
Check the key and return the resource that has to be unlocked. Only one resource may be unlocked at one time.	
<ul style="list-style-type: none"> > 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 5-29 XcpAppl_Unlock

5.5.6 XcpAppl_CalibrationWrite

uint8 XcpAppl_CalibrationWrite (Xcp_AddressPtrType Dst, uint8 *Src, uint8 Size)	
Dst	Destination address as integer.
Src	Pointer to source of data.
Size	Size of data to copy from Src to Dst.
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
Check addresses for valid write access and copy data from source to destination.	
<ul style="list-style-type: none"> > 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 5-30 XcpAppl_CalibrationWrite

5.5.7 XcpAppl_MeasurementRead

uint8 XcpAppl_MeasurementRead(uint8 *Dst, Xcp_AddressPtrType Src, uint8 Size)	
Dst	Pointer to destination address
Src	Source address of data as integer
Size	Size of data to copy from Src to Dst.
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
Check addresses for valid read access and copy data from source to destination.	
<ul style="list-style-type: none"> > 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 5-31 XcpAppl_MeasurementRead

5.5.8 XcpAppl_CheckReadAccess

uint8 XcpAppl_CheckReadAccess(Xcp_ChannelType XcpChannel, Xcp_AddressPtrType Address, uint32 Size)	
XcpChannel	The channel number in multi client mode.
Address	Destination address to check.
Size	Size of data to check.
uint8	XCP_CMD_DENIED : if access is denied XCP_CMD_OK : if access is granted
Check addresses for valid read access.	

- > 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 5-32 XcpAppl_CheckReadAccess

5.5.9 XcpAppl_CheckProgramAccess

uint8 **XcpAppl_CheckProgramAccess**(Xcp_AddressPtrType Address, uint32 Size)

Address	Destination address to check.
---------	-------------------------------

Size	Size of data to check.
------	------------------------

uint8	XCP_CMD_DENIED : if access is denied XCP_CMD_OK : if access is granted
-------	---

Check addresses for valid write flash access.

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

Expected Caller Context

- > Task level

Table 5-33 XcpAppl_CheckProgramAccess

5.5.10 XcpAppl_UserService

uint8 **XcpAppl_UserService**(uint8 *Cmd)

Cmd	Pointer to command string
-----	---------------------------

uint8	XCP_CMD_OK : if command is accepted. XCP_CMD_PENDING : if command is performed asynchronously. XCP_CMD_SYNTAX : if command is not accepted.
-------	---

Application specific user command.
<ul style="list-style-type: none">> 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 5-34 XcpAppl_UserService

5.5.11 XcpAppl_OpenCmdIf

<pre>uint8 XcpAppl_OpenCmdIf(Xcp_ChannelType XcpChannel, uint8 *Cmd, uint8 *Response, uint8 *Length)</pre>	
XcpChannel	The channel number in multi client mode.
Cmd	Pointer to command string
Response	Pointer to response string
Length	Pointer to response length

uint8 XcpAppl_SendStall (Xcp_ChannelType XcpChannel)	
XcpChannel	The channel number in multi client mode.
uint8	0 : Reject sending of new message. 1 : continue processing.
Resolve a transmit stall condition in Xcp_Putchar or Xcp_SendEvent.	
<ul style="list-style-type: none"> > 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 5-36 XcpAppl_SendStall

5.5.13 XcpAppl_DisableNormalOperation

uint8 XcpAppl_DisableNormalOperation (Xcp_AddressPtrType Address, uint16 Size)	
Address	Address (where the flash kernel is downloaded to)
Size	Size (of the flash kernel)
uint8	XCP_CMD_OK: download of flash kernel confirmed XCP_CMD_DENIED: download of flash kernel refused
Prior to the flash kernel download has the ECU's normal operation to be stopped in order to avoid misbehavior due to data inconsistencies.	
<ul style="list-style-type: none"> > 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 5-37 XcpAppl_DisableNormalOperation

5.5.14 XcpAppl_StartBootLoader

uint8 XcpAppl_StartBootLoader(void)	
-	-
uint8	<div>This function should not return. XCP_CMD_OK : positive response XCP_CMD_BUSY : negative response</div>
Start of the boot loader.	
<div>> 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</div>	
Expected Caller Context	
> Task level	

Table 5-38 XcpAppl_StartBootLoader

5.5.15 XcpAppl_Reset

Expected Caller Context

- > Task level

Table 5-39 XcpAppl_Reset

5.5.16 XcpAppl_ProgramStart

```
uint8 XcpAppl_ProgramStart( void )
```

-	-
---	---

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

Prepare the ECU for flash programming.

- > 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 5-40 XcpAppl_ProgramStart

5.5.17 XcpAppl_FlashClear

```
uint8 XcpAppl_FlashClear( uint8 *Address, uint32 Size )
```

Address	Address of memory area to clear
---------	---------------------------------

Size	Size of memory area to clear
------	------------------------------

uint8	XCP_CMD_OK : Flash memory erase done XCP_CMD_PENDING : Call Xcp_SendCrm() when done XCP_CMD_ERROR : Flash memory erase error
-------	--

Clear the flash memory, before the flash memory will be reprogrammed.

- > 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 5-41 XcpAppl_FlashClear

5.5.18 XcpAppl_FlashProgram

```
uint8 XcpAppl_FlashProgram( const uint8 *Data, uint8 *Address, uint8 Size )
```

Data	Pointer to data.
------	------------------

Address	Address of memory to store data at.
---------	-------------------------------------

Size	Size of data.
------	---------------

uint8	XCP_CMD_OK : Flash memory programming finished XCP_CMD_PENDING : Flash memory programming in progress. Xcp_SendCrm has to be called when done.
-------	--

Program the cleared flash memory.

- > 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 5-42 XcpAppl_FlashProgram

5.5.19 XcpAppl_DaqResume

```
uint8 XcpAppl_DaqResume( Xcp_ChannelType XcpChannel, Xcp_ChannelStruct *Channel )
```

XcpChannel	The channel number in multi client mode.
Channel	Pointer to dynamic DAQ list structure
uint8	Boolean flag whether valid DAQ list was restored.
<p>Resume the automatic data transfer.</p> <p>The whole dynamic DAQ list structure that had been stored in non-volatile memory within the service <code>XcpAppl_DaqResumeStore(. .)</code> has to be restored to RAM.</p>	
<ul style="list-style-type: none"> > 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 5-43 XcpAppl_DaqResume

5.5.20 XcpAppl_DaqResumeStore

<pre>void XcpAppl_DaqResumeStore(Xcp_ChannelType XcpChannel, const Xcp_ChannelStruct *Channel, uint8 MeasurementStart)</pre>	
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
-	-
<p>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.</p> <p>After a cold start or reset the dynamic DAQ list structure has to be restored by the application callback service <code>XcpAppl_DaqResume(. .)</code> when the flag <code>MeasurementStart</code> is > 0.</p>	

- > 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 5-44 XcpAppl_DaqResumeStore

5.5.21 XcpAppl_DaqResumeClear

```
void XcpAppl_DaqResumeClear( Xcp_ChannelType XcpChannel )
```

XcpChannel	The channel number in multi client mode.
------------	--

-	-
---	---

The whole dynamic DAQ list structure that had been stored in non-volatile memory within the service `XcpAppl_DaqResumeStore(..)` has to be cleared.

- > 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 5-45 XcpAppl_DaqResumeClear

5.5.22 XcpAppl_CalResumeStore

```
boolean XcpAppl_CalResumeStore( Xcp_ChannelType XcpChannel )
```

XcpChannel	The channel number in multi client mode.
------------	--

boolean	If true the calibration page was stored.
---------	--

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.

- > 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 5-46 XcpAppl_CalResumeStore

5.5.23 XcpAppl_GetCalPage

```
uint8 XcpAppl_GetCalPage( uint8 Segment, uint8 Mode )
```

Segment	Logical data segment number
Mode	Access mode The access mode can be one of the following values: 1 : ECU access 2 : XCP access

uint8	Logical data page number
-------	--------------------------

This function returns the logical number of the calibration data page that is currently activated for the specified access mode and data segment.

- > 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

Expected Caller Context

- > Task level

Table 5-47 XcpAppl_GetCalPage

5.5.24 XcpAppl_SetCalPage

uint8 XcpAppl_SetCalPage (uint8 Segment, uint8 Page, uint8 Mode)	
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.
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
Switch pages, e.g. from reference page to working page.	
<ul style="list-style-type: none"> > This function is asynchronous if it returns XCP_CMD_PENDING. > 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 	
Expected Caller Context	
> Task level	

Table 5-48 XcpAppl_SetCalPage

5.5.25 XcpAppl_CopyCalPage

uint8 XcpAppl_CopyCalPage (uint8 SrcSeg, uint8 SrcPage, uint8 DestSeg, uint8 DestPage)	
SrcSeg	Source segment.
SrcPage	Source page.
DestSeg	Destination segment.
DestPage	Destination page.

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.
Copying of calibration data pages. The pages are copied from source to destination.	
<ul style="list-style-type: none">> This function is asynchronous if it returns XCP_CMD_PENDING.> 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/XcpCopyPage	
Expected Caller Context	
<ul style="list-style-type: none">> Task level	

Table 5-49 XcpAppl_CopyCalPage

5.5.26 XcpAppl_SetFreezeMode

void XcpAppl_SetFreezeMode (uint8 Segment, uint8 Mode)	
Segment	Segment to set freeze mode
Mode	New freeze mode
–	–
Setting the freeze mode of a certain segment. Application must store the current freeze mode of each segment.	
<ul style="list-style-type: none">> 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	
<ul style="list-style-type: none">> Task level	

Table 5-50 XcpAppl_SetFreezeMode

5.5.27 XcpAppl_GetFreezeMode

uint8 XcpAppl_GetFreezeMode(uint8 Segment)	
Segment	Segment to read freeze mode
uint8	Return the current freeze mode, set by XcpAppl_SetFreezeMode().
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.	
<ul style="list-style-type: none">> 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 5-51 XcpAppl_GetFreezeMode

5.5.28 XcpAppl_CalculateChecksum

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

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

Expected Caller Context

- > Task level

Table 5-52 XcpAppl_CalculateChecksum

5.5.29 XcpAppl_ConStateNotification

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

XcpChannel	The channel number in multi client mode.
ConnectionState	The new connection state (<code>XCP_CON_STATE_RESUME</code> , <code>XCP_CON_STATE_DISCONNECTED</code> , <code>XCP_CON_STATE_CONNECTED</code>).

–	–
---	---

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

- > 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 5-53 XcpAppl_ConStateNotification

5.5.30 XcpAppl_MemCpy

```
uint8 XcpAppl_MemCpy( uint8 * Dst, const uint8 * Src, uint16 Size )
```

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.

-	-
Copies data from source to destination.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is non-reentrant. > The XCP Protocol Layer has to be initialized correctly. 	
Expected Caller Context	
<ul style="list-style-type: none"> > Task and interrupt level 	

Table 5-54 XcpAppl_MemCpy

5.6 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.

DET	Det_ReportError
OS	GetCoreID

Table 5-55 Services used by the XCP

6

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

7.1 Abbreviations

	File Extension for an SAM MC language File
	SAM 2 eta language
	pplication rogramming nterface
	ssociation for tandardization of utomation and easuring Systems
	assing
	ontroller rea etwork
	ibration
	Systems
	o man
	ommand ransfer bject
	Synchronous ata c uision
	ata ength ode (Number of data bytes of a CAN message)
	ata ink ayer
	ata ransfer bject
	lectronic ontrol nit
	or Packet
	ent packet
	Identifies a CAN message
	nterrupt ervice outine
	aster alibration ystem
	One or more signals are assigned to each message.
	bject escriptor able
	iginal quipment anufacturer (vehicle manufacturer)

	nified ata rotocol / nternet rotocol
	niversal erial us
	Universal Measurement and alibration rotocol
	ector nformatik GmbH

Table 7-1 Abbreviations

8

Visit our website for more information on

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

www.vector.com