

XCP Protocol Layer

Technical Reference

Version 2.05.00

Version:	2.05.00
Status:	Released

1 History

Date	Version	Remarks
2005-01-17	1.00.00	ESCAN00009143: Initial draft Warning Text added
2005-06-22	1.01.00	FAQ extended: ESCAN00012356, ESCAN00012314 ESCAN00012617: Add service to retrieve XCP state
2005-12-20	1.02.00	ESCAN00013883: Revise Resume Mode
2006-03-09	1.03.00	ESCAN00015608: Support command TRANSPORT_LAYER_CMD ESCAN00015609: Support XCP on FlexRay Transport Layer
2006-04-24	1.04.00	ESCAN00015913: Correct filenames Data page banking support of application callback template added
2006-05-08	1.05.00	ESCAN00016263: Describe support of reflected CRC16 CCITT ESCAN00016159: Add demo disclaimer to XCP Basic
2006-05-29	1.06.00	ESCAN00016226: Support XCP on LIN Transport Layer
2006-07-20	1.07.00	ESCAN00012636: Add configuration with GENy ESCAN00016956: Support AUTOSAR CRC module
2006-10-26	1.08.00	ESCAN00018115: DPRAM Support only available in XCP Basic ESCAN00017948: Add paging support ESCAN00017221: Documentation of reentrant capability of all functions
2007-01-18	1.09.00	ESCAN00018809: Support data paging on Star12X / Cosmic
2007-05-07	1.10.00	Description of new features added
2007-09-14	1.11.00	Segment freeze mode now supported
2008-07-23	1.12.00	ESCAN00028586: Support of Program_Start callback ESCAN00017955: Support MIN_ST_PGM ESCAN00017952: Open Interface for command processing
2008-09-10	1.13.00	Additional pending return value of call backs added MIN_ST configuration added
2008-12-01	1.14.00	ESCAN00018157: SERV_RESET is not supported ESCAN00032344: Update of XCP Basic Limitations
2009-05-14	1.15.00	ESCAN00033909: New features implemented: Prog Write Protection, Timestamps, Calibration activation
2009-07-30	1.15.01	Fixed some editorial errors
2009-11-13	1.16.00	Added AUTOSAR Compiler Abstraction
2010-04-30	1.16.01	Fixed some editorial errors
2010-07-27	1.16.02	Fixed some editorial errors
2010-08-19	1.17.00	ESCAN00044693: New callbacks XcpCalibrationWrite and XcpCalibrationRead ESCAN00042867: Support Multiple Transport Layers

2010-12-10	1.18.00	ESCAN00045981: Add support to read out FR Parameters
2011-07-20	1.19.00	ESCAN00049542: Describe IDT_VECTOR_MAPNAMES format in TechRef ESCAN00043487: XCP shall support user selectable behaviour of Send Queue overrun
2011-08-04		ESCAN00052564: Adapt ReadCcConfig Parameter to ASR3.2.1
2012-02-20	1.19.01	ESCAN00055214: DAQ Lists can be extended after START_STOP_SYNCH
2012-09-03	1.19.02	ESCAN00061159: Provide an API to detect XCP state and usage
2012-11-08	1.19.03	Added Option for AMD Runtime Measurement
2011-03-23	2.00.00	ESCAN00049471: Create branch for AUTOSAR 4
2013-02-11	2.01.01	Editorial Changes
2013-07-08	2.02.00	ESCAN00068035: Xcp_SetTransmissionMode not supported ESCAN00070127: AR4-322/AR3_2552: Support of Vx1000 System ESCAN00070082: The API <u>ApplXcpDaqResumeStore</u> has a wrong description ESCAN00069019: Mapping to critical sections not described in detail for Protocol Layer ESCAN00068639: Describe data consistency on ODT Level ESCAN00067332: Document the usage of the <u>Xcp_MainFunction/XcpBackground</u>
2013-12-04	2.03.00	ESCAN00072401: Support custom CRC <u>Cbk</u> ESCAN00072326: Support Generic GET_ID
2014-08-15	2.03.01	ESCAN00077231: AR3-2679: Description BCD-coded return-value of Xcp_GetVersionInfo() in TechRef ESCAN00077813: Specify supported ASAM Version
2015-02-02	2.03.02	ESCAN00080981: SET_CAL_PAGE is limited to synchronous operation
2015-06-09	2.04.00	ESCAN00082215: FEAT-1450: Basic <u>MultiCore</u> XCP
2016-02-18	2.04.01	ESCAN00087492: New API ApplXcpMeasurementRead/ApplXcpCalibrationWrite not documented ESCAN00087496: Return code description of <u>ApplXcp</u> call-backs incomplete
2016-10-04	2.05.00	Replaced Xcp_Control API by variable. ESCAN00091747: FEAT-1980: Add Multi Client / Multi Connection support ESCAN00092229: Support API to modify Protection State

**Please note**

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.

**Note for XCP Basic**

Please note, that the demo and example programs only show special aspects of the software. With regard to the fact that these programs are meant for demonstration purposes only, Vector Informatik's liability shall be expressly excluded in cases of ordinary negligence, to the extent admissible by law or statute.

Contents

1	History	2
2	Overview	11
2.1	Abbreviations and Items used in this paper	11
2.2	Naming Conventions	13
3	Functional Description	14
3.1	Overview of the Functional Scope	14
3.2	Communication Mode Info	14
3.3	Block Transfer Communication Model (XCP Professional only)	14
3.4	Slave Device Identification	14
3.4.1	XCP Station Identifier	14
3.4.2	XCP Generic Identification	15
3.4.3	Identification of FlexRay Parameters	15
3.5	Seed & Key	15
3.6	Checksum Calculation	17
3.6.1	Custom CRC calculation	17
3.7	MainFunction	17
3.8	Memory Protection (XCP Professional only)	18
3.9	Memory Access by Application	18
3.9.1	Special use case "Type Safe Copy"	18
3.10	Event Codes	18
3.11	Service Request Messages	19
3.12	User Defined Command	19
3.13	Transport Layer Command	19
3.14	Synchronous Data Transfer	20
3.14.1	Synchronous Data Acquisition (DAQ)	20
3.14.2	DAQ Timestamp	20
3.14.3	Power-Up Data Transfer	21
3.14.4	Send Queue	21
3.14.5	Data Stimulation (STIM)	22
3.14.6	Bypassing	22
3.14.7	Data Acquisition Plug & Play Mechanisms	22
3.14.8	Event Channel Plug & Play Mechanism	23
3.14.9	Data consistency	23
3.15	The Online Data Calibration Model	24
3.15.1	Page Switching	24
3.15.2	Page Switching Plug & Play Mechanism	24
3.15.3	Calibration Data Page Copying	24

3.15.4	Freeze Mode Handling	24
3.16	Flash Programming	25
3.16.1	Flash Programming by the ECU's Application	25
3.16.2	Flash Programming with a Flash Kernel	26
3.16.3	Flash Programming Write Protection	26
3.17	EEPROM Access	26
3.18	Parameter Check	27
3.19	Performance Optimizations	27
3.20	Interrupt Locks / Exclusive Areas	27
3.20.1	XCP_EXCLUSIVE_AREA_0	28
3.20.2	XCP_EXCLUSIVE_AREA_1	28
3.20.3	XCP_EXCLUSIVE_AREA_2	28
3.21	Basic Multi Core support	28
3.21.1	Type safe copy	28
3.22	Accessing internal data	28
3.23	En- / Disabling the XCP module	28
3.24	XCP measurement during the follow up time	29
4	Integration into the Application	30
4.1	Files of XCP Professional	30
4.2	Version changes	30
4.3	Compiler Abstraction and Memory Mapping	30
4.4	Support of Vx1000 Integration	31
5	Feature List	32
6	Description of the API	34
6.1	Version of the Source Code	34
6.2	XCP Services called by the Application	35
6.2.1	Xcp_InitMemory: Initialization of the XCP Protocol Layer Memory ...	35
6.2.2	Xcp_Init: Initialization of the XCP Protocol Layer	35
6.2.3	Xcp_Event: Handling of a data acquisition event channel	36
6.2.4	Xcp_StimEventStatus: Check data stimulation events	37
6.2.5	Xcp_MainFunction: Background calculation of checksum	37
6.2.6	Xcp_SendEvent: Transmission of event codes	38
6.2.7	Xcp_Putchar: Put a char into a service request packet	38
6.2.8	Xcp_Print: Transmission of a service request packet	39
6.2.9	Xcp_Disconnect: Disconnect from XCP master	40
6.2.10	Xcp_SendCrm: Transmit response or error packet	40
6.2.11	Xcp_GetXcpDataPointer: Request internal data pointer	41
6.2.12	Xcp_GetVersionInfo: Request module version information	41

6.2.13	Xcp_ModifyProtectionStatus: Influence seed&key behaviour	42
6.3	XCP Protocol Layer Functions, called by the XCP Transport Layer.....	42
6.3.1	Xcp_Command: Evaluation of XCP packets and command interpreter	43
6.3.2	Xcp_SendCallBack: Confirmation of the successful transmission of a XCP packet.....	43
6.3.3	Xcp_GetSessionStatus: Get session state of XCP	44
6.3.4	Xcp_SetActiveTI: Set the active Transport Layer.....	45
6.3.5	Xcp_GetActiveTI: Get the currently active Transport Layer	45
6.4	XCP Transport Layer Services called by the XCP Protocol Layer	46
6.4.1	<Bus>Xcp_Send: Request for the transmission of a DTO or CTO message	46
6.4.2	<Bus>Xcp_SendFlush: Flush transmit buffer	46
6.4.3	XcpAppl_InterruptEnable: Enable interrupts.....	47
6.4.4	XcpAppl_InterruptDisable: Disable interrupts	48
6.4.5	<Bus>Xcp_TLService: Transport Layer specific commands.....	48
6.5	Application Services called by the XCP Protocol Layer	49
6.5.1	XcpAppl_GetPointer: Pointer conversion	49
6.5.2	XcpAppl_GetIdData: Get Identification.....	50
6.5.3	XcpAppl_GetSeed: Generate a seed	50
6.5.4	XcpAppl_Unlock: Valid key and unlock resource.....	51
6.5.5	XcpAppl_CheckReadEEPROM: Check read access from EEPROM	52
6.5.6	XcpAppl_CheckWriteEEPROM: Check write access to the EEPROM	53
6.5.7	XcpAppl_CheckWriteAccess: Check address for valid write access. 53	
6.5.8	XcpAppl_CheckReadAccess: Check address for valid read access. 54	
6.5.9	XcpAppl_CheckDAQAccess: Check address for valid read or write access.....	55
6.5.10	XcpAppl_CheckProgramAccess: Check address for valid write access.....	55
6.5.11	XcpAppl_UserService: User defined command.....	56
6.5.12	XcpAppl_OpenCmdIf: XCP command extension interface	56
6.5.13	XcpAppl_SendStall: Resolve a transmit stall condition.....	57
6.5.14	XcpAppl_DisableNormalOperation: Disable normal operation of the ECU	58
6.5.15	XcpAppl_StartBootLoader: Start of boot loader.....	58
6.5.16	XcpAppl_Reset: Perform ECU reset	59
6.5.17	XcpAppl_ProgramStart: Prepare flash programming.....	59
6.5.18	XcpAppl_FlashClear: Clear flash memory	60
6.5.19	XcpAppl_FlashProgram: Program flash memory.....	60
6.5.20	XcpAppl_DaqResume: Resume automatic data transfer.....	61
6.5.21	XcpAppl_DaqResumeStore: Store DAQ lists for resume mode.....	62

6.5.22	XcpAppl_DaqResumeClear: Clear stored DAQ lists.....	62
6.5.23	XcpAppl_CalResumeStore: Store Calibration data for resume mode.....	63
6.5.24	XcpAppl_GetTimestamp: Returns the current timestamp	64
6.5.25	XcpAppl_GetCalPage: Get calibration page.....	64
6.5.26	XcpAppl_SetCalPage: Set calibration page	65
6.5.27	XcpAppl_CopyCalPage: Copying of calibration data pages	66
6.5.28	XcpAppl_SetFreezeMode: Setting the freeze mode of a segment....	66
6.5.29	XcpAppl_GetFreezeMode: Reading the freeze mode of a segment .	67
6.5.30	XcpAppl_Read: Read a single byte from memory	67
6.5.31	XcpAppl_Write: Write a single byte to RAM.....	68
6.5.32	XcpAppl_MeasurementRead: Read multiple bytes from memory.....	68
6.5.33	XcpAppl_CalibrationWrite: Write multiple bytes to memory	69
6.5.34	XcpAppl_ReadChecksumValue: Read checksum value	70
6.5.35	XcpAppl_CalculateChecksum: Custom checksum calculation	70
6.6	XCP Protocol Layer Functions that can be overwritten.....	71
6.6.1	Xcp_MemCpy: Copying of a memory range	71
6.6.2	Xcp_MemSet: Initialization of a memory range	72
6.6.3	Xcp_MemClr: Clear a memory range	72
6.7	AUTOSAR CRC Module Services called by the XCP Protocol Layer (XCP Professional Only).....	73
6.8	Configuration without Generation Tool	75
6.8.1	Compiler Switches	75
6.8.2	Configuration of Constant Definitions	78
6.8.3	Configuration of the CPU Type.....	80
6.8.4	Configuration of Slave Device Identification	80
6.8.5	Configuration of the Event Channel Plug & Play Mechanism	82
6.8.6	Configuration of the DAQ Time Stamped Mode.....	83
6.8.7	Configuration of the Flash Programming Plug & Play Mechanism....	84
6.8.8	Configuration of the Page Switching Plug & Play Mechanism	85
6.8.9	Configuration of the used Transport Layer	85
7	Resource Requirements.....	87
8	Limitations	88
8.1	General Limitations	88
8.2	Limitations Regarding Platforms, Compilers and Memory Models.....	89
9	FAQ.....	90
9.1	Invalid Time Stamp Unit	90
9.2	Support of small and medium memory model	90
9.3	Small memory model on ST10 / XC16X / C16X with Tasking Compiler	91

9.4 Data Page Banking on Star12X / Metrowerks 91

9.5 Memory model banked on Star12X / Cosmic 91

9.6 Reflected CRC16 CCITT Checksum Calculation Algorithm 92

10 Bibliography..... 93

11 Contact..... 94

Illustrations

Figure 3-1 Data consistency 23

2 Overview

This document describes the features, API, configuration and integration of the XCP Protocol Layer. Both XCP versions: XCP Professional and XCP Basic are covered by this document. Chapters that are only relevant for XCP Professional are marked.

This document does not cover the XCP Transport Layers for CAN, FlexRay and LIN, which are available at Vector Informatik.

Please refer to [IV] for further information about XCP on CAN and the integration of XCP on CAN with the Vector CANbedded software components. Further information about XCP on FlexRay Transport Layer and XCP on LIN Transport Layer can be found in its 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. Referred functions are always shown in the single channel mode.



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.

2.1 Abbreviations and Items used in this paper

Abbreviations	Complete expression
A2L	File Extension for an ASAM 2MC Language File
AML	ASAM 2 Meta Language
API	Application Programming Interface
ASAM	Association for Standardization of Automation and Measuring Systems
BYP	BYPassing
CAN	Controller Area Network
CAL	CALibration
CANape	Calibration and Measurement Data Acquisition for Electronic Control Systems

CMD	Command
CTO	Command Transfer Object
DAQ	Synchronous Data Acquisition
DLC	Data Length Code (Number of data bytes of a CAN message)
DLL	Data link layer
DTO	Data Transfer Object
ECU	Electronic Control Unit
ERR	Error Packet
EV	Event packet
ID	Identifier (of a CAN message)
Identifier	Identifies a CAN message
ISR	Interrupt Service Routine
MCS	Master Calibration System
Message	One or more signals are assigned to each message.
ODT	Object Descriptor Table
OEM	Original equipment manufacturer (vehicle manufacturer)
PAG	PAGing
PID	Packet Identifier
PGM	Programming
RAM	Random Access Memory
RES	Command Response Packet
ROM	Read Only Memory
SERV	Service Request Packet
STIM	Stimulation
TCP/IP	Transfer Control Protocol / Internet Protocol
UDP/IP	Unified Data Protocol / Internet Protocol
USB	Universal Serial Bus
XCP	Universal Measurement and Calibration Protocol
VI	Vector Informatik GmbH

Also refer to 'AN-AND-1-108 Glossary of CAN Protocol Terminology.pdf', which can be found in the download area of <http://www.vector-informatik.de>.

2.2 Naming Conventions

The names of the access functions provided by the XCP Protocol Layer always start with a prefix that includes the characters `xcp`. The characters `xcp` are surrounded by an abbreviation which refers to the service or to the layer which requests a XCP service. The designation of the main services is listed below:

Naming conventions

<code>Xcp_...</code>	<p>It is mandatory to use all functions beginning with <code>Xcp...</code>. These services are called by either the data link layer or the application. They are e.g. used for the initialization of the XCP Protocol Layer and for the cyclic background task.</p>
<code>XcpAppl_...</code>	<p>The functions, starting with <code>ApplXcp...</code> are functions that are provided either by any XCP Transport Layer or the application and are called by the XCP Protocol Layer.</p> <p>These services are user callback functions that are application specific and have to be implemented depending on the application.</p>

3 Functional Description

3.1 Overview of the Functional Scope

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.

3.2 Communication Mode Info

In order to gather information about the XCP Slave device, e.g. the implementation version number of the XCP Protocol Layer and supported communications models, the communication mode info can be enabled by the switch `XCP_ENABLE_COMM_MODE_INFO`.

3.3 Block Transfer Communication Model (XCP Professional only)

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.

Refer to chapter 6.8.1 for configuration details.

3.4 Slave Device Identification

3.4.1 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.

Refer to chapter 6.8.4.1 (Identification by ASAM-MC2 Filename without Path and Extension) for configuration details.

3.4.2 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. The following function

```
uint32 XcpAppl_GetIdData( MTABYTEPTR *pData, uint8 id ) (6.5.2)
```

has to set a pointer to the identification information based on the requested id and return the length of this information.

Refer to chapter 6.8.4.2 (Automatic Session Configuration with MAP Filenames) for an example implementation.

3.4.3 Identification of FlexRay Parameters

If the “Virtual FlexRay Parameters” feature is enabled, the parameters can be read out in a platform independent way. They will be provided as virtual measurement values that can be read at fixed memory locations with a configurable Address Extension.

To calculate the memory address for each parameter please read the Technical Reference and the AUTOSAR specification of the FlexRay Driver. Each FlexRay parameter is defined with a unique ID to be used as parameter for the API call. Use this ID and multiply it with four to get the address where this variable can be measured at.

If this parameter is enabled the API:

```
Std_ReturnType FrIf_ReadCCConfig( uint8 ClusterIdx, uint8  
FrIf_CCLLPParamIndex, P2VAR(uint32, AUTOMATIC, FRIF_APPL_DATA)  
FrIf_CCLLPParamValue )
```

will be called. The FlexRay parameters can be measured from CAN and FlexRay but the API is only provided if the FlexRay Interface is present.

3.5 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.

The seed & key functionality can be enabled with the switch XCP_ENABLE_SEED_KEY and disabled with XCP_DISABLE_SEED_KEY in order to save ROM. Also refer to chapter 6.8.1.

The application callback function

```
uint8 XcpAppl_GetSeed( uint8 Xcp_Channel, MEMORY_ROM uint8  
resourceMask, BYTEPTR seed ) (6.5.3)
```

returns a seed that is transferred to the MCS. The callback function

```
uint8 XcpAppl_Unlock( uint8 Xcp_Channel, MEMORY_ROM uint8 *key,  
MEMORY_ROM uint8 length ) (6.5.4)
```

has to verify a received key and if appropriate return the resource that shall be unlocked.

The service:

```
uint8  Xcp_ModifyProtectionStatus (  uint8  Xcp_Channel,  uint8
andState, uint8 orState )
```

(6.2.13)

can be used to modify the protection state by software.

Annotation for the usage of CANape

The calculation of the key is done in a DLL named SEEDKEY1.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.

Example Implementation for SEEDKEY1.DLL

The function call of ASAP1A_XCP_ComputeKeyFromSeed() is standardized by the ASAM committee.



Example

```
FILE SEEDKEY1.H
#ifdef _SEEDKEY_H_

#define _SEEDKEY_H_
#ifdef DllImport
#define DllImport __declspec(dllimport)
#endif
#ifdef DllExport
#define DllExport __declspec(dllexport)
#endif
#ifdef SEEDKEYAPI_IMPL
#define SEEDKEYAPI DllExport __cdecl
#else
#define SEEDKEYAPI DllImport __cdecl
#endif
#ifdef __cplusplus
extern "C" {
#endif

BOOL SEEDKEYAPI ASAP1A_XCP_ComputeKeyFromSeed( BYTE *seed,
    unsigned short sizeSeed,
    BYTE *key,
    unsigned short maxSizeKey,
    unsigned short *sizeKey
);
#ifdef __cplusplus
}
#endif
#endif

FILE SEEDKEY1.C
#include <windows.h>
#define SEEDKEYAPI_IMPL
#include "SeedKey1.h"

extern "C" {
BOOL SEEDKEYAPI ASAP1A_XCP_ComputeKeyFromSeed( BYTE *seed,
    unsigned short sizeSeed,
```



```

BYTE *key,
unsigned short maxSizeKey,
unsigned short *sizeKey
)
{ // in that example sizeSeed == 4 is expected only
  if( sizeSeed != 4 ) return FALSE;
  if( maxSizeKey < 4 ) return FALSE;
  *((unsigned long*)key) *= 3;
  *((unsigned long*)key) &= 0x55555555;
  *((unsigned long*)key) *= 5;
  *sizeKey = 4;
  return TRUE;
}

```

3.6 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.

XCP Professional allows the usage of the AUTOSAR CRC Module [VII]. If the AUTOSAR CRC Module is used also the XCP CRC32 algorithm can be used.

Also refer to 6.8.2.1 'Table of Checksum Calculation Methods'.

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

3.6.1 Custom CRC calculation

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

```

uint8 XcpAppl_CalculateChecksum( uint8 Xcp_Channel, ROMBYTEPTR
pMemArea, BYTEPTR pRes, uint32 length )

```

is called. 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 every case the response frame has to be assembled.

3.7 MainFunction

The Xcp provides a MainFunction:

```
void Xcp_MainFunction( void ) (6.2.5)
```

which 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.

3.8 Memory Protection (XCP Professional only)

If `XCP_ENABLE_WRITE_PROTECTION` is defined write access of specific RAM areas can be checked with the function

```
uint8 XcpAppl_CheckWriteAccess( MTABYTEPTR addr, uint8 size ) (6.5.7)
```

It should only be used, if write protection of memory areas is required.

If `XCP_ENABLE_READ_PROTECTION` is defined read access of specific RAM areas can be checked with the function

```
uint8 XcpAppl_CheckReadAccess( MTABYTEPTR addr, uint8 size ) (6.5.8)
```

It should only be used, if read protection of memory areas is required.

While the first two functions are used during polling, the following function is used for DAQ/STIM access:

```
uint8 XcpAppl_CheckDAQAccess( DAQBYTEPTR addr, uint8 size ) (6.5.9)
```

These functions can be used to protect memory areas that are not allowed to be accessed, e.g. memory mapped registers or the xcp memory itself.

3.9 Memory Access by Application

There are two APIs available that allow memory access by application. Those APIs can be enabled by setting `XCP_ENABLE_CALIBRATION_MEM_ACCESS_BY_APPL`. Please note that these API are only used for polling access. DAQ/STIM still uses direct memory access.

```
uint8 XcpAppl_CalibrationWrite( P2VAR(void, AUTOMATIC, XCP_APPL_DATA) dst, P2CONST(void, AUTOMATIC, XCP_APPL_DATA) src, uint8 len ) (6.5.33)
```

```
uint8 XcpAppl_MeasurementRead( P2VAR(void, AUTOMATIC, XCP_APPL_DATA) dst, P2CONST(void, AUTOMATIC, XCP_APPL_DATA) src, uint8 len ) (6.5.32)
```

If the option `XCP_ENABLE_DAQ_MEM_ACCESS_BY_APPL` is set the function `XcpAppl_MeasurementRead` is also called for DAQ measurement.

3.9.1 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. See 3.21.1 for further information.

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 `XCP_ENABLE_SEND_EVENT`. The transmission is done by the service

```
void Xcp_SendEvent( uint8 evc, ROMBYTEPTR c, uint8 len ) (6.2.6)
```

The event codes can be found in the following table.

Event	Code	Description
EV_RESUME_MODE	0x00	The slave indicates that it is starting in RESUME mode.
EV_CLEAR_DAQ	0x01	The slave indicates that the DAQ configuration in non-volatile memory has been cleared.
EV_STORE_DAQ	0x02	The slave indicates that the DAQ configuration has been stored into non-volatile memory.
EV_STORE_CAL	0x03	The slave indicates that the calibration data has been stored.
EV_CMD_PENDING	0x05	The slave requests the master to restart the time-out detection.
EV_DAQ_OVERLOAD	0x06	The slave indicates an overload situation when transferring DAQ lists.
EV_SESSION_TERMINATED	0x07	The slave indicates to the master that it autonomously decided to disconnect the current XCP session.
EV_USER	0xFE	User-defined event.
EV_TRANSPORT	0xFF	Transport layer specific event.

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 due to these packets are not being acknowledged.

The service request messages can be sent by the following functions

```
void Xcp_PutChar ( const uint8 c ) (6.2.7)
```

```
void Xcp_Print ( const uint8 *str ) (6.2.8)
```

Refer to 6.8.1 for the configuration of the service request message.

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 `XCP_ENABLE_USER_COMMAND` and upon reception of the user command the following callback function is called by the XCP command processor:

```
uint8 XcpAppl_UserService ( uint8 Xcp_Channel, ROMBYTEPTR pCmd (6.5.11)
                             )
```

3.13 Transport Layer Command

The transport layer commands are received by the XCP Protocol Layer and processed by the XCP Transport Layer. The XCP Protocol Layer transmits the XCP response packets (RES) or XCP error packets (ERR).

The transport layer command is enabled by setting `XCP_ENABLE_TL_COMMAND`. Upon reception of any transport layer command the following callback function is called by the XCP command processor:

```
uint8 ApplXcpTlService ( ROMBYTEPTR pCmd ) (6.4.5)
```

3.14 Synchronous Data Transfer

3.14.1 Synchronous Data Acquisition (DAQ)

The synchronous data transfer can be enabled with the compiler switch `XCP_ENABLE_DAQ`. 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 cyclically 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, which is defined in the MCS. Note that the event channel numbers are given by the GenTool when the Event Info feature is used.

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.

3.14.2 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:

```
XcpDaqTimestampType XcpAppl_GetTimestamp ( void ) (6.5.24)
```

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 increment value of this timer must be configured in the configuration Tool (e.g. GENy) accordingly.

For the configuration of the DAQ time stamped mode refer to chapter 6.8.6 (Configuration of the DAQ Time Stamped Mode).

3.14.3 Power-Up Data Transfer

Power-up data transfer (also called resume mode) allows automatic data transfer (DAQ, STIM) 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.

```
uint8 XcpAppl_DaqResume ( uint8 Xcp_Channel, tXcpDaq * daq ) (6.5.20)
```

```
void XcpAppl_DaqResumeStore ( uint8 Xcp_Channel, const tXcpDaq  
    * daq ) (6.5.21)
```

```
void XcpAppl_DaqResumeClear ( uint8 Xcp_Channel ) (6.5.22)
```

```
uint8 XcpAppl_CalResumeStore ( uint8 Xcp_Channel ) (6.5.23)
```

To use the resume mode the compiler switches `XCP_ENBALE_Daq` and `XCP_ENABLE_RESUME_MODE` have to be defined.

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 Master Tool the before mentioned call-back `XcpAppl_DaqResumeStore` is called by the `MainFunction`.

```
void Xcp_MainFunction( void ) (6.2.5)
```

Annotation for the usage of CANape

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

3.14.4 Send Queue

The send queue is used to store measurement values until they can be transmitted on the bus. This is required if the used Transport Layer does not perform buffering on its own. Vector Transport Layers do not buffer any data and therefore this feature should be used.

The send queue size can be indirectly configured in the GenTool. It is defined by the parameter "Memory Size" – the memory size used by the dynamic DAQ lists. As the DAQ lists are created during runtime by the tool no detailed calculation is possible. A worst case analysis can be made and the parameter should be chosen such that enough space is left for the send queue.

Furthermore the behaviour of the send queue in case of an overrun condition can be influenced. There are two possible options:

1. Throw away oldest element

→ The oldest odt in the send queue is discarded and the new measurement value is inserted. The send queue behaves as a ring buffer.

2. Throw away latest element

→ The latest measurement values are discarded. The send queue behaves like a linear buffer.

The GenTool option “Replace First Element” determines the default behaviour. The behaviour can be changed during runtime by modifying the variable `xcp.Daq.SendQueueBehaviour`. If this variable is zero linear mode is selected, if this variable is one the ring buffer mode is selected. This variable can be modified by the Master Tool.

3.14.5 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 the compiler switches `XCP_ENBALE_DAO` and `XCP_ENABLE_STIM` have to be defined.

3.14.6 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.

3.14.7 Data Acquisition Plug & Play Mechanisms

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

- > general information on the DAQ processor
(enabled with `XCP_ENABLE_DAO_PROCESSOR_INFO`)
- > general information on DAQ processing resolution
(enabled with `XCP_ENABLE_DAO_RESOLUTION_INFO`)

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

3.14.8 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.

Please refer to chapter 6.8.5 (Configuration of the Event Channel Plug & Play Mechanism) for details about the configuration of this plug & play mechanism.

Annotation for the usage of CANape

If the plug & play mechanism is not built-in, you must open the dialog XCP Device Setup with the menu command Tools|Driver parameters. Go to the Event tab. Make one entry for each event channel. An event channel is an `Xcp_Event(x)` function call in ECU source code.

3.14.9 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()`

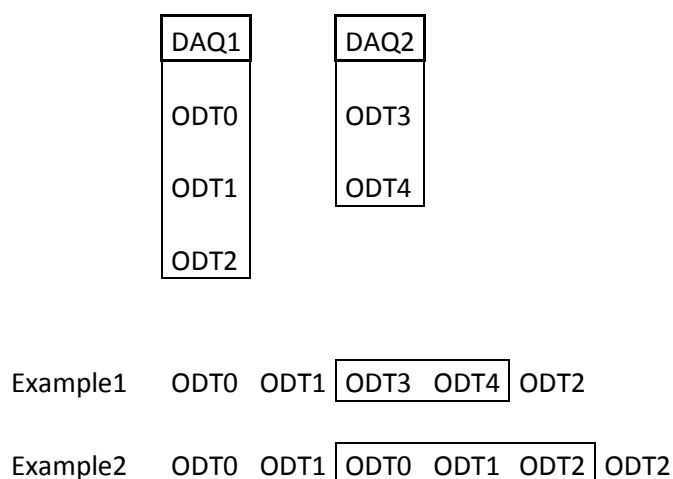


Figure 3-1 Data consistency

3.15 The Online Data Calibration Model

3.15.1 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 XCP_ENABLE_CALIBRATION_PAGE definition.

The following application callback functions have to be implemented:

```
uint8 XcpApp1_GetCalPage ( uint8 Xcp_Channel, uint8 segment,
                          uint8 mode )                               (6.5.25)
```

```
uint8 XcpApp1_SetCalPage ( uint8 Xcp_Channel, uint8 segment,
                          uint8 page, uint8 mode )                 (6.5.26)
```

Annotation for the usage 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.

3.15.2 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

Also refer to chapter 6.8.8 (Configuration of the Page Switching Plug & Play Mechanism) and to the XCP Specification [II].

The page switching plug & play mechanism is enabled with the switch XCP_ENABLE_PAGE_INFO.

3.15.3 Calibration Data Page Copying

Calibration data page copying is performed by the XCP command COPY_CAL_PAGE. To enable this feature the compiler switch XCP_ENABLE_PAGE_COPY has to be set.

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

```
uint8 XcpApp1_CopyCalPage( uint8 Xcp_Channel, uint8 srcSeg,
                          uint8 srcPage, uint8 destSeg, uint8
                          destPage )                               (6.5.27)
```

3.15.4 Freeze Mode Handling

Freeze mode handling is performed by the XCP commands SET_SEGMENT_MODE and GET_SEGMENT_MODE. To enable this feature the compiler switch XCP_ENABLE_PAGE_FREEZE has to be set.

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

```
void XcpApp1_SetFreezeMode( uint8 segment, uint8 mode )          (6.5.28)
```

```
uint8 XcpApp1_GetFreezeMode( uint8 segment )                    (6.5.29)
```


3.16 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.

3.16.1 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_FORMAT¹, PROGRAM_VERIFY¹.

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

```
uint8 XcpAppl_ProgramStart( void ) (6.5.17)
```

```
uint8 XcpAppl_FlashClear( MTABYTEPTR a, uint32 size ) (6.5.18)
```

```
uint8 XcpAppl_FlashProgram( ROMBYTEPTR data,  
                             MTABYTEPTR a, uint8 size ) (6.5.19)
```

The flash programming is enabled with the switch XCP_ENABLE_PROGRAM.

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.

3.16.1.1 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

Also refer to chapter 6.8.7 (Configuration of the Flash Programming Plug & Play Mechanism) and to the XCP Specification [II].

The flash programming plug & play mechanism is enabled with the switch XCP_ENABLE_PROGRAM_INFO.

¹ Command not supported

3.16.2 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

```
uint8 XcpAppl_DisableNormalOperation( MTABYTEPTR a, uint16 size
                                     )                                     (6.5.14)
```

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 `XCP_ENABLE_BOOTLOADER_DOWNLOAD`.

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.

3.16.3 Flash Programming Write Protection

If `XCP_ENABLE_PROGRAMMING_WRITE_PROTECTION` is defined write access of specific FLASH areas can be checked with the function

```
uint8 XcpAppl_CheckProgramAccess
      ( MTABYTEPTR addr, uint32 size )                                     (6.5.10)
```

It should only be used, if write protection of flash areas is required.

3.17 EEPROM Access

For uploading data from the ECU to a MCS the XCP commands `SHORT_UPLOAD` and `UPLOAD` are used. The switch `XCP_ENABLE_READ_EEPROM` allows EEPROM access for these commands.

Before reading from an address it is checked within the following callback function whether EEPROM or RAM is accessed:

```
uint8 XcpAppl_CheckReadEEPROM
      ( MTABYTEPTR addr, uint8 size, BYTEPTR data )                     (6.5.5)
```

The EEPROM access is directly performed within this function.

For downloading data from the MCS to the ECU the XCP commands `SHORT_DOWNLOAD`, `DOWNLOAD`, `DOWNLOAD_NEXT` and `DOWNLOAD_MAX` can be used. The switch `XCP_ENABLE_WRITE_EEPROM` allows the EEPROM access for these commands.

Also before writing to an address within the following callback function it is checked whether EEPROM or RAM is accessed

```
uint8 XcpAppl_CheckWriteEEPROM
( uint8 Xcp_Channel, MTABYTEPTR addr, uint8 size,
  ROMBYTEPTR data )
```

 (6.5.6)

3.18 Parameter Check

As long as the XCP Protocol Layer is not thoroughly tested together with the XCP Transport Layer and the application, the parameter check should be enabled. This is done by setting the compiler switch `XCP_ENABLE_PARAMETER_CHECK`.

The parameter check may be removed in order to save code space.

3.19 Performance Optimizations

The XCP Protocol Layer is a platform comprehensive higher software layer and therefore platform specific optimizations are not implemented. However it is possible to apply platform specific optimizations.

The following memory access functions can be overwritten by either macros or functions:

```
void Xcp_MemCpy( DAQBYTEPTR dest,
                 ROMDAQBYTEPTR src, uint16 n )
```

 (6.6.1)

```
void Xcp_MemSet( BYTEPTR p, uint16 n, uint8 b )
```

 (6.6.2)

```
static void Xcp_MemClr( BYTEPTR p, uint16 n )
```

 (6.6.3)

It is recommended to use DMA access as far as possible for faster execution of these services.

3.20 Interrupt Locks / Exclusive Areas

The functions `Xcp_Event`, `Xcp_SendCallBack`, `Xcp_MainFunction` and `Xcp_Command` are not reentrant. If one of these functions may interrupt one of the others, they must be protected against each other. See also 3.14.9.

For this purpose the Xcp Protocol Layer makes use of three exclusive areas. The SchM must provide 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:

3.20.1 XCP_EXCLUSIVE_AREA_0

Is used by functions `Xcp_SendCallBack`, `Xcp_MainFunction` and `Xcp_Command` to protect these non-reentrant functions.

3.20.2 XCP_EXCLUSIVE_AREA_1

Is used by `Xcp_Event` during DAQ measurement.

3.20.3 XCP_EXCLUSIVE_AREA_2

Is used by `Xcp_Event` during STIM measurement.

3.21 Basic Multi Core support

3.21.1 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.

With this option disabled, 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 Master Tools perform an optimization by grouping measurement values. This option must be disabled, otherwise they do not represent unique data types anymore.

3.22 Accessing internal data

The function

```
void Xcp_GetXcpDataPointer (P2VAR(tXcpData, AUTOMATIC,  
XCP_APPL_DATA) *pXcpData ) (6.2.11)
```

provides access to the internal data structure of the XCP module. By means of this function the internal data can be preset to a certain value. This can be used to process a measurement further that has been started in application mode but is finished in boot mode.

As the whole data can be accessed, it must be handled with care.

3.23 En- / Disabling the XCP module

The variable `Xcp_ControlState`

can be used to en- or disable the XCP module during run time. Thus the XCP functionality can be controlled by the application.

Furthermore two macros are available: `XCP_ACTIVATE` and `XCP_DEACTIVATE`. They can be used to control the protocol and transport layer together, i.e. enabling or disabling

them as a whole. It is recommended to use these macros. It is also recommended to perform a `Xcp_Disconnect()` API call to bring the Xcp in a save state before it is disabled.

3.24 XCP measurement during the follow up 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:

`SessionStatusType Xcp_GetSessionStatus (void)` (6.3.3)

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



Example

```
{
    SessionStatusType sessionState;

    sessionState = Xcp_GetSessionStatus();
    if( 0 != (sessionState & SS_CONNECTED) )
    {
        /* Is the xcp actively used? */
        if( 0 != (sessionState & (SS_DAQ | SS_POLLING)) )
        {
            /* Yes, reload timer */
            swTimer = XCP_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.

4 Integration into the Application




This chapter describes the steps for the integration of the XCP Protocol Layer into an application environment of an ECU.

4.1 Files of XCP Professional

The XCP Protocol Layer consists of the following files.

Files of the XCP Protocol Layer		
<code>Xcp.c</code>	XCP Professional source code. This file must not be changed by the user!	
<code>Xcp.h</code>	API of XCP Professional. This file must not be changed by the user!	
<code>_xcp_appl.c</code>	Template that contains the application callback functions of the XCP Protocol Layer. It is just an example and has to be customized.	
<code>v_def.h</code>	General Vector definitions of memory qualifiers and types. This file must not be changed by the application!	

Additionally the following files are generated by the generation tool. If no generation tool or if CANgen is used the XCP Protocol Layer has to be customized manually. In this case the following files will be available as template.

Files generated by GENy		
<code>xcp_Cfg.h</code>	XCP Protocol Layer configuration file.	
<code>xcp_Lcfg.c</code>	Parameter definition for the XCP Protocol Layer.	
<code>xcp_Lcfg.h</code>	External declarations for the parameters.	

Note that all files of XCP Professional **must not** be changed manually!

4.2 Version changes

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

4.3 Compiler Abstraction and Memory Mapping

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

The following table contains the memory section names and the compiler abstraction definitions defined for XCP, and illustrates their assignment among each other.

Memory Mapping Sections	Compiler Abstraction Definitions				
	XCP_CONST	XCP_DAQ_DATA	XCP_MTA_DATA	XCP_APPL_DATA	XCP_CODE
XCP_START_SEC_CONST_16BIT	■				
XCP_START_SEC_CONST_8BIT	■				
XCP_START_SEC_VAR_NOINIT_UNSPECIFIED				■	
XCP_START_SEC_VAR_NOINIT_8BIT				■	
XCP_START_SEC_CODE					■
XCP_START_SEC_VAR_INIT_UNSPECIFIED_SAFE				■	

Table 4-1 Compiler abstraction and memory mapping

Please see the document: “AUTOSAR_SWS_CompilerAbstraction.pdf” for details about how to use these definitions.

4.4 Support of Vx1000 Integration

The XcpProf provides basic support for the Vx1000 Hardware which can be enabled in the configuration tool. If enabled the code size is increased, yet the same API calls as used for the XcpProf are reused for the Vx which minimizes integration effort.

When the option is enabled the sources provided with your Vx1000 hardware must be integrated. The XcpProf includes the Vx1000.h header and makes use of the respective macros.

If the Vx hardware is attached prior to ECU Initialization the XcpProf itself is deactivated, hence no access via the bus interface is possible anymore. If you want to perform measurement & calibration via the bus interface again, detach the Vx hardware and perform an ECU reset.

5 Feature List

This general feature list describes the overall feature set of the XCP Protocol Layer.

Description of the XCP functionality	Functions
Initialization	
Initialization	Xcp_Init ApplXcpInit
Task	
Background task	Xcp_MainFunction
XCP Command Processor	
Command Processor	Xcp_Command
Transmission and Confirmation of XCP Packets	<Bus>Xcp_Send Xcp_SendCallBack
Transmission of Response packets	Xcp_SendCrm
Transmission of XCP Packets	XcpAppl_SendStall <Bus>Xcp_SendFlush
XCP Commands	
Get Identification	XcpAppl_GetIdData
Seed & Key	XcpAppl_GetSeed XcpAppl_Unlock
Short Download	-
Modify Bits	-
Write DAQ Multiple	XcpAppl_CheckDAQAccess
Transport Layer Command	<Bus>Xcp_TLService
Open Command Interface	XcpAppl_OpenCmdIf
User command	XcpAppl_UserService
Data Acquisition (DAQ)	
Synchronous Data Acquisition and Stimulation	Xcp_Event XcpAppl_CheckDAQAccess
DAQ Timestamp	XcpAppl_GetTimestamp
Resume Mode	XcpAppl_DaqResume XcpAppl_DaqResumeStore XcpAppl_DaqResumeClear XcpAppl_CalResumeStore
Online Data Calibration	
Calibration page switching	XcpAppl_GetCalPage XcpAppl_SetCalPage
Copy calibration page	XcpAppl_CopyCalPage
Freeze Mode	XcpAppl_SetFreezeMode XcpAppl_GetFreezeMode
Boot loader Download	

Disable normal operation of ECU	XcpAppl_DisableNormalOperation
Start of the boot loader	XcpAppl_StartBootLoader
Flash Programming	
Reset of ECU	XcpAppl_Reset
Clear flash memory	XcpAppl_FlashClear
Prepare flash programming	XcpAppl_ProgramStart
Program flash memory	XcpAppl_FlashProgram
Special Features	
Interrupt Control	ApplXcpInterruptEnable ApplXcpInterruptDisable
Event Codes	Xcp_SendEvent
Service Request Packets	Xcp_Putchar Xcp_Print
Disconnect XCP	Xcp_Disconnect
Pointer conversion	XcpAppl_GetPointer
EEPROM access	XcpAppl_CheckReadEEPROM XcpAppl_CheckWriteEEPROM
Write protection	XcpAppl_CheckWriteAccess
Read protection	XcpAppl_CheckReadAccess
Overwriteable macros	Xcp_MemCpy Xcp_MemSet Xcp_MemClr Xcp_SendDto
Access to internal data	Xcp_GetXcpDataPointer
En-/Disable Calibration	-
Programming Write Protection	XcpAppl_CheckProgramAccess
Session Status	Xcp_GetSessionStatus

6 Description of the API

The XCP Protocol Layer application programming interface consists of services, which are realized by function calls. These services are called wherever they are required. They transfer information to- or take over information from the XCP Protocol Layer. This information is stored in the XCP Protocol Layer until it is not required anymore, respectively until it is changed by other operations.

Examples for calling the services of the XCP Protocol Layer can be found in the description of the services.

6.1 Version of the Source Code

The source code version of the XCP Protocol Layer is provided by three BCD coded constants:

```
CONST(uint8, XCP_CONST) kXcpMainVersion =  
(uint8) (CP_XCP_VERSION >> 8);  
  
CONST(uint8, XCP_CONST) kXcpSubVersion =  
(uint8) (CP_XCP_VERSION);  
  
CONST(uint8, XCP_CONST) kXcpReleaseVersion =  
(uint8) (CP_XCP_RELEASE_VERSION);
```



Example

Version 1.00.00 is registered as:

```
kXcpMainVersion      = 0x01;  
kXcpSubVersion       = 0x00;  
kXcpReleaseVersion   = 0x00;
```

These constants are declared as external and can be read by the application at any time.

Alternatively the Version can be obtained with the `GetVersionInfo` API if enabled:

```
void Xcp_GetVersionInfo (P2VAR(Std_VersionInfoType, AUTOMATIC,  
XCP_APPL_DATA) XcpVerInfoPtr) (6.2.12)
```

6.2 XCP Services called by the Application

The following XCP services that are called by the application are all not reentrant. If they are called within interrupt context at least the CAN-Interrupts have to be disabled.

6.2.1 Xcp_InitMemory: Initialization of the XCP Protocol Layer Memory

Xcp_InitMemory

Prototype	
Single Channel	
Single Receive Channel	void Xcp_InitMemory (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
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	
<ul style="list-style-type: none"> > Call context: Task and interrupt level > This service function has to be called after the initialization of XCP Transport Layer. > 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 before. 	

6.2.2 Xcp_Init: Initialization of the XCP Protocol Layer

Xcp_Init

Prototype	
Single Channel	
Single Receive Channel	void Xcp_Init (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
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.

Particularities and Limitations

- > Call context: Task and interrupt level
- > This service function has to be called after the initialization of XCP Transport Layer.
- > 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 before.

6.2.3 Xcp_Event: Handling of a data acquisition event channel

Xcp_Event

Prototype

Single Channel

Single Receive Channel uint8 **Xcp_Event** (uint8 event)

Multi Channel

Indexed not supported

Code replicated not supported

Parameter

event	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
-------	-----------------------------------------------------------------------------------------------------------------------------------

Return code

uint8	XCP_EVENT_NO : Inactive (DAQ not running, Event not configured) XCP_EVENT_DAQ : DAQ active */ XCP_EVENT_DAQ_OVERRUN : DAQ queue overflow 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 (0); /* 10ms cycle */
xcp_Event (1); /* 100ms cycle */
xcp_Event (2); /* Crank synchronous cycle */
```

Particularities and Limitations

- > The XCP Protocol Layer has been initialized correctly and XCP is in connected state.
- > Data acquisition has to be enabled: XCP_ENABLE_DAQ has to be defined
- > Call context: Task and interrupt level (not reentrant)

6.2.4 Xcp_StimEventStatus: Check data stimulation events

Xcp_StimEventStatus

Prototype

Single Channel

Single Receive Channel uint8 **Xcp_StimEventStatus** (uint8 event, uint8 action)

Multi Channel

Indexed not supported

Code replicated not supported

Parameter

event	Event channel number
action	STIM_CHECK_ODT_BUFFER : check ODT buffer STIM_RESET_ODT_BUFFER : reset ODT buffer

Return code

uint8	0 : stimulation data not available 1 : new stimulation data is available
-------	-----------------------------------------------------------------------------

Functional Description

Check if data stimulation (STIM) event can perform or delete the buffers.

Particularities and Limitations

- > The XCP Protocol Layer has been initialized correctly and XCP is in connected state.
- > Data acquisition has to be enabled: XCP_ENABLE_STIM has to be defined
- > Call context: Task and interrupt level (not reentrant)

6.2.5 Xcp_MainFunction: Background calculation of checksum

Xcp_MainFunction

Prototype

Single Channel

Single Receive Channel void **Xcp_MainFunction** (void)

Multi Channel

Indexed not supported

Code replicated not supported

Parameter

-	-
---	---

Return code	
uint8	0 : background calculation finished 1 : background calculation is still in progress
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 <code>Xcp_MainFunction</code> . Therefore <code>Xcp_MainFunction</code> should be called periodically whenever the ECU's CPU is idle.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly > Call context: Task level 	

6.2.6 Xcp_SendEvent: Transmission of event codes

Xcp_SendEvent

Prototype	
Single Channel	
Single Receive Channel	void Xcp_SendEvent (uint8 Xcp_Channel, uint8 evc, ROMBYTEPTR c, uint8 len)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
evc	event code
c	pointer to event data
len	event data length
Return code	
-	-
Functional Description	
Transmission of event codes via event packets (EV). Please refer to chapter 3.10 Event Codes.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Data acquisition has to be enabled: XCP_ENABLE_SEND_EVENT has to be defined > Call context: Task and interrupt level 	

6.2.7 Xcp_Putchar: Put a char into a service request packet

Xcp_Putchar

Prototype	
Single Channel	

Single Receive Channel	void Xcp_Putchar (uint8 Xcp_Channel, const uint8 c)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
c	character that is put in a service request packet
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 out in the service request packet.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > The switch <code>XCP_ENABLE_SERV_TEXT_PUTCHAR</code> has to be defined > Call context: Task and interrupt level (not reentrant) 	

6.2.8 Xcp_Print: Transmission of a service request packet

Xcp_Print

Prototype	
Single Channel	
Single Receive Channel	void Xcp_Print (uint8 Xcp_Channel, P2CONST(uint8, AUTOMATIC, XCP_APPL_DATA) str)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
str	pointer to a string that is terminated by 0x00
Return code	
-	-
Functional Description	
Transmission of a service request packet (SERV). The string <code>str</code> is sent via service request packets. The string has to be terminated by 0x00.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > The switch <code>XCP_ENABLE_SERV_TEXT_PRINT</code> has to be defined > Call context: Task and interrupt level (not reentrant) 	

6.2.9 Xcp_Disconnect: Disconnect from XCP master

Xcp_Disconnect

Prototype	
Single Channel	
Single Receive Channel	void Xcp_Disconnect (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
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	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Call context: Task and interrupt level (not reentrant) 	

6.2.10 Xcp_SendCrm: Transmit response or error packet

Xcp_SendCrm

Prototype	
Single Channel	
Single Receive Channel	void Xcp_SendCrm (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Transmission of a command response packet (RES), or error packet (ERR) if no other packet is pending.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly, XCP is in connected state and a command packet (CMD) has been received. > Call context: Task and interrupt level (not reentrant) 	

6.2.11 Xcp_GetXcpDataPointer: Request internal data pointer

Xcp_GetXcpDataPointer

Prototype	
Single Channel	
Single Receive Channel	void Xcp_GetXcpDataPointer (tXcpData ** pXcpData)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
pXcpData	pointer to store the pointer to the module internal data
Return code	
-	-
Functional Description	
With this function the pointer to the module internal data can be received. With this pointer the internal variable can be set to a certain configuration (e.g. after entering a boot mode where no connection shall be established again). As this pointer allows the access to all internal data it must be handled with care.	
Particularities and Limitations	
➤ The switch <code>XCP_ENABLE_GET_XCP_DATA_POINTER</code> has to be defined	

6.2.12 Xcp_GetVersionInfo: Request module version information

Xcp_GetVersionInfo

Prototype	
Single Channel	
Single Receive Channel	void Xcp_GetVersionInfo (P2VAR(Std_VersionInfoType, AUTOMATIC, XCP_APPL_DATA) XcpVerInfoPtr)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
XcpVerInfoPtr	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

- The switch `XCP_ENABLE_VERSION_INFO_API` has to be defined
- > Call context: task level (Re-entrant)

6.2.13 Xcp_ModifyProtectionStatus: Influence seed&key behaviour

Xcp_ModifyProtectionStatus

Prototype

Single Channel

Single Receive Channel	void Xcp_ModifyProtectionStatus (uint8 Xcp_Channel, uint8 andState, uint8 orState)
------------------------	---------------------------------------------------------------------------------------------

Multi Channel

Indexed	not supported
Code replicated	not supported

Parameter

Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
andState	The following flags: <code>RM_CAL_PAG</code> , <code>RM_DAQ</code> , <code>RM_STIM</code> and <code>RM_PGM</code> can be used to clear the protection state of the respective resource. The modified state is persistent until <code>Xcp_Init</code> .
orState	The following flags: <code>RM_CAL_PAG</code> , <code>RM_DAQ</code> , <code>RM_STIM</code> and <code>RM_PGM</code> can be used to set the protection state of the respective resource. The modified state is persistent until <code>Xcp_Init</code> .

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

- The switch `XCP_ENABLE_VERSION_INFO_API` has to be defined
- > Call context: task level (Re-entrant)

6.3 XCP Protocol Layer Functions, called by the XCP Transport Layer

For using the following functions there are some limitations which have to be taken into consideration – especially when using an operation system like, i.e. OSEK OS:

- > The ISR level for the transmission and reception of CAN messages has to be the same.
- > Interrupts must be mutually
- > No nested calls of these functions are allowed. (i.e. these functions are not reentrant)

All functions provided by the application must match the required interfaces. This can be ensured by including the header file in the modules which provide the required functions. If these interfaces do not match unexpected run-time behavior may occur.

6.3.1 Xcp_Command: Evaluation of XCP packets and command interpreter

Xcp_Command

Prototype	
Single Channel	
Single Receive Channel	void Xcp_Command (uint8 Xcp_Channel, P2CONST(uint32, AUTOMATIC, XCP_APPL_DATA) pCommand)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
pCommand	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	
<ul style="list-style-type: none"> > The XCP Protocol Layer has to be initialized correctly. > Call context: Task and interrupt level (not reentrant) 	

6.3.2 Xcp_SendCallBack: Confirmation of the successful transmission of a XCP packet

Xcp_SendCallBack

Prototype	
Single Channel	
Single Receive Channel	uint8 Xcp_SendCallBack (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-

Return code	
uint8	0 : if the XCP Protocol Layer is idle (no transmit messages are pending)
Functional Description	
<p>The XCP Protocol Layer does not call <Bus>Xcp_Send again, until Xcp_SendCallback has confirmed the successful transmission of the previous message. Xcp_SendCallback transmits pending data acquisition messages by calling <Bus>Xcp_Send again.</p> <p>Note that if Xcp_SendCallback is called from inside <Bus>Xcp_Send a recursion occurs, which assumes enough space on the call stack.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly. > Call context: Task and interrupt level (not reentrant) 	

6.3.3 Xcp_GetSessionStatus: Get session state of XCP

Xcp_GetSessionStatus

Prototype	
Single Channel	
Single Receive Channel	SessionStatusType Xcp_GetSessionStatus (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
SS_CONNECTED	XCP is connected
SS_DAQ	DAQ measurement is running
SS_POLLING	Polling is running (depending on polling rate this flag is not always set)
Functional Description	
<p>This service can be used to get the session state of the XCP Protocol Layer. The session state is returned as bit mask where the individual bits can be tested.</p> <p>E.g. this service is used by the XCP on CAN Transport Layer to determine the connection state in case multiple CAN channels are used and can be used by the application to prevent an ECU shutdown.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has to be initialized correctly. > Call context: Task and interrupt level (not reentrant) > Enabled/Disabled by XCP_XXX_GET_SESSION_STATUS_API 	

6.3.4 Xcp_SetActiveTI: Set the active Transport Layer

Xcp_SetActiveTI

Prototype	
Single Channel	
Single Receive Channel	void Xcp_SetActiveTI (uint8 Xcp_Channel, uint8 MaxCto, uint8 MaxDto, uint8 ActiveTI)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
MaxCto	Max CTO used by the respective XCP Transport Layer
MaxDto	Max DTO used by the respective XCP Transport Layer
ActiveTI	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	
Set the active Transport Layer the XCP Protocol Layer uses. 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	
<ul style="list-style-type: none"> > The XCP Protocol Layer has to be initialized correctly. > Call context: Task and interrupt level (not reentrant) 	

6.3.5 Xcp_GetActiveTI: Get the currently active Transport Layer

Xcp_GetActiveTI

Prototype	
Single Channel	
Single Receive Channel	uint8 Xcp_GetActiveTI (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
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

Get the active Transport Layer the XCP Protocol Layer uses.

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

- > The XCP Protocol Layer has to be initialized correctly.
- > Call context: Task and interrupt level (not reentrant)

6.4 XCP Transport Layer 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 component's header.

6.4.1 <Bus>Xcp_Send: Request for the transmission of a DTO or CTO message

<Bus>Xcp_Send

Prototype

Single Channel

Single Receive Channel	void <Bus>Xcp_Send (uint8 Xcp_Channel, uint8 len, ROMBYTEPTR msg)
------------------------	----------------------------------------------------------------------------------

Multi Channel

Indexed	not supported
Code replicated	not supported

Parameter

len	Length of message data
msg	Pointer to message

Return code

uint8	0 : if the XCP Protocol Layer is idle (no transmit messages are pending)
-------	--------------------------------------------------------------------------

Functional Description

Requests for the transmission of a command transfer object (CTO) or data transfer object (DTO). Xcp_SendCallback must be called after the successful transmission of any XCP message. The XCP Protocol Layer will not request further transmissions, until Xcp_SendCallback has been called.

Particularities and Limitations

- > Call context: Task and interrupt level (not reentrant)
- > <Bus>Xcp_Send is not defined as macro

6.4.2 <Bus>Xcp_SendFlush: Flush transmit buffer

<Bus>Xcp_SendFlush

Prototype	
Single Channel	
Single Receive Channel	void <Bus>Xcp_SendFlush (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Flush the transmit buffer.	
Particularities and Limitations	
-	

6.4.3 XcpAppl_InterruptEnable: Enable interrupts

XcpAppl_InterruptEnable

Prototype	
Single Channel	
Single Receive Channel	void XcpAppl_InterruptEnable (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Enabling of the global interrupts.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly > Call context: Task and interrupt level > This function is reentrant! > The function XcpAppl_InterruptEnable can be overwritten by the macro XcpAppl_InterruptEnable. 	

6.4.4 XcpAppl_InterruptDisable: Disable interrupts

XcpAppl_InterruptDisable

Prototype	
Single Channel	
Single Receive Channel	void XcpAppl_InterruptDisable (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Disabling of the global interrupts.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly > Call context: Task and interrupt level > This function is reentrant! > The function <code>XcpAppl_InterruptDisable</code> can be overwritten by the macro <code>XcpAppl_InterruptDisable</code>. 	

6.4.5 <Bus>Xcp_TLService: Transport Layer specific commands

<Bus>Xcp_TLService

Prototype	
Single Channel	
Single Receive Channel	uint8 <Bus>Xcp_TLService (uint8 Xcp_Channel, ROMBYTEPTR pCmd)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
pCmd	Pointer to COMMAND that has been received by the XCP Slave.
Return code	
uint8	<div style="display: flex; justify-content: space-between;"> <div> XCP_CMD_OK : XCP_CMD_PENDING : XCP_CMD_SYNTAX : XCP_CMD_BUSY : XCP_CMD_UNKNOWN : XCP_CMD_OUT_OF_RANGE : </div> <div> Done Call Xcp_SendCrm() when done Error not executed not implemented optional command command parameters out of range </div> </div>

Functional Description

Transport Layer specific command that is processed within the XCP Transport Layer.

Particularities and Limitations

- > XCP is initialized correctly
- > Call context: Task and interrupt level
- > The switch `XCP_ENABLE_TL_COMMAND` has to be defined

6.5 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 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.

6.5.1 XcpAppl_GetPointer: Pointer conversion

XcpAppl_GetPointer

Prototype

Single Channel

Single Receive Channel	MTABYTEPTR XcpAppl_GetPointer (uint8 addr_ext, uint32 addr)
------------------------	----------------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

addr_ext	8 bit address extension
----------	-------------------------

addr	32 bit address
------	----------------

Return code

MTABYTEPTR	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 for memory transfers like DOWNLOAD and UPLOAD.

Example:

The following code shows an example of a typical implementation of `XcpAppl_GetPointer`:

```
MTABYTEPTR XcpAppl_GetPointer( uint8 addr_ext, uint32 addr )
{
    return (MTABYTEPTR)addr;
}
```

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > This function can be overwritten by defining `XcpAppl_GetPointer` as macro.

6.5.2 XcpAppl_GetIdData: Get Identification

XcpAppl_GetIdData

Prototype

Single Channel

Single Receive Channel	uint32 XcpAppl_GetIdData (MTABYTEPTR *pData, uint8 id)
------------------------	-----------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

pData	Returns a pointer to a pointer of MAP file names
id	Identification of the requested information/identification

Return code

uint32	length of the MAP file names
--------	------------------------------

Functional Description

Returns a pointer to a pointer of MAP file names.
Refer to chapter 3.4.2 (XCP Generic Identification).

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > The switch `XCP_ENABLE_GET_ID_GENERIC` has to be defined

6.5.3 XcpAppl_GetSeed: Generate a seed

XcpAppl_GetSeed

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_GetSeed (uint8 Xcp_Channel, const uint8 resource, P2VAR(uint8, AUTOMATIC, XCP_APPL_DATA) seed)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
Resource	Resource for which the seed has to be generated <u>XCP Professional and XPC Basic</u> RM_CAL_PAG : to unlock the resource calibration/paging RM_DAQ : to unlock the resource data acquisition <u>XCP Professional only</u> RM_STIM : to unlock the resource stimulation RM_PGM : to unlock the resource programming
Seed	Pointer to RAM where the seed has to be generated to.
Return code	
uint8	The length of the generated seed that is returned by <i>seed</i> .
Functional Description	
Generate a seed for the appropriate resource. The seed has a maximum length of MAX_CTO-2 bytes.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch <code>XCP_ENABLE_SEED_KEY</code> has to be defined 	

6.5.4 XcpAppl_Unlock: Valid key and unlock resource

XcpAppl_Unlock

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_Unlock (uint8 Xcp_Channel, P2CONST(uint8, AUTOMATIC, XCP_APPL_DATA) key, const uint8 length)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
key	Pointer to the key.

length	Length of the key.
Return code	
uint8	<u>XCP Professional and XPC Basic</u> 0 : if the key is not valid RM_CAL_PAG : to unlock the resource calibration/paging RM_DAQ : to unlock the resource data acquisition <u>XCP Professional only</u> RM_STIM : to unlock the resource stimulation RM_PGM : to unlock the resource programming
Functional Description	
Check the key and return the resource that has to be unlocked. Only one resource may be unlocked at one time.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch XCP_ENABLE_SEED_KEY has to be defined	

6.5.5 XcpAppl_CheckReadEEPROM: Check read access from EEPROM

XcpAppl_CheckReadEEPROM

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_CheckReadEEPROM (MTABYTEPTR addr, uint8 size, BYTEPTR data)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
addr	Address that is checked
size	Number of bytes
data	Pointer to data (if the address is on the EEPROM the data is written here)
Return code	
uint8	XCP_CMD_OK : EEPROM read XCP_CMD_DENIED : This is not EEPROM XCP_CMD_PENDING : EEPROM read in progress, call Xcp_SendCrm when done
Functional Description	
Checks whether the address lies within the EEPROM memory or in the RAM area. If the area is within the EEPROM area <code>size</code> data byte are read from <code>addr</code> and written to <code>data</code> .	

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > The switch `XCP_ENABLE_READ_EEPROM` has to be defined

6.5.6 XcpAppl_CheckWriteEEPROM: Check write access to the EEPROM

XcpAppl_CheckWriteEEPROM

Prototype

Single Channel

Single Receive Channel	uint8 XcpAppl_CheckWriteEEPROM (uint8 Xcp_Channel, MTABYTEPTR addr, uint8 size, ROMBYTEPTR data)
------------------------	----------------------------------------------------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
addr	Address that is checked
size	number of bytes
data	pointer to data (if addr is on the EEPROM this data is written to addr)

Return code

uint8	<code>XCP_CMD_OK</code> : EEPROM written <code>XCP_CMD_DENIED</code> : This is not EEPROM <code>XCP_CMD_PENDING</code> : EEPROM write in progress, call <code>XcpSendCrm</code> when done
-------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Functional Description

Checks whether the address `addr` is within the EEPROM memory. If not, the function returns `XCP_CMD_DENIED`. If it lies within, EEPROM programming is performed. The function may return during programming with `XCP_CMD_PENDING` or may wait until the programming sequence has finished and then returns with `XCP_CMD_OK`.

If the programming sequence has finished, the `Xcp_SendCrm` function must be called.

`Xcp_SendCrm` is an internal function of the XCP Protocol Layer.

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > The switch `XCP_ENABLE_WRITE_EEPROM` has to be defined

6.5.7 XcpAppl_CheckWriteAccess: Check address for valid write access

XcpAppl_CheckWriteAccess

Prototype

Single Channel

Single Receive Channel	uint8 XcpAppl_CheckWriteAccess (MTABYTEPTR address, uint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
address	address
size	number of bytes
Return code	
uint8	XCP_CMD_DENIED : if access is denied XCP_CMD_OK : if access is granted
Functional Description	
Check addresses for valid write access. A write access is enabled with the XCP_ENABLE_WRITE_PROTECTION, it should be only used, if write protection of memory areas is required	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch XCP_ENABLE_WRITE_PROTECTION has to be defined > Can be overwritten by the macro XcpAppl_CheckWriteAccess 	

6.5.8 XcpAppl_CheckReadAccess: Check address for valid read access

XcpAppl_CheckReadAccess

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_CheckReadAccess (MTABYTEPTR address, uint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
address	address
size	number of bytes
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. A read access is enabled with the XCP_ENABLE_READ_PROTECTION, it should be only used, if read protection of memory areas is required	

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > The switch `XCP_ENABLE_READ_PROTECTION` has to be defined
- > Can be overwritten by the macro `XcpAppl_CheckReadAccess`

6.5.9 XcpAppl_CheckDAQAccess: Check address for valid read or write access

XcpAppl_CheckDAQAccess

Prototype

Single Channel

Single Receive Channel	uint8 XcpAppl_CheckDAQAccess (DAQBYTEPTR address, uint8 size)
------------------------	------------------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

address	address
size	number of bytes

Return code

uint8	<code>XCP_CMD_DENIED</code> : if access is denied <code>XCP_CMD_OK</code> : if access is granted
-------	-----------------------------------------------------------------------------------------------------

Functional Description

Check addresses for valid read or write access. This callback is called when a `WRITE_DAQ` command is performed. Therefore it is not possible to know whether this is a read or write access. Out of this reason this unified function is called.

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > The switch `XCP_ENABLE_READ_PROTECTION` or `XCP_ENABLE_WRITE_PROTECTION` has to be defined

6.5.10 XcpAppl_CheckProgramAccess: Check address for valid write access

XcpAppl_CheckProgramAccess

Prototype

Single Channel

Single Receive Channel	uint8 XcpAppl_CheckProgramAccess (MTABYTEPTR address, uint32 size)
------------------------	-----------------------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

address	address
---------	---------

size	number of bytes
Return code	
uint8	XCP_CMD_DENIED : if access is denied XCP_CMD_OK : if access is granted
Functional Description	
Check addresses for valid write access. A write access is enabled with the XCP_ENABLE_PROGRAMMING_WRITE_PROTECTION, it should be only used, if write protection of memory areas is required	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch XCP_ENABLE_PROGRAMMING_WRITE_PROTECTION has to be defined > Can be overwritten by the macro XcpAppl_CheckWriteAccess 	

6.5.11 XcpAppl_UserService: User defined command

XcpAppl_UserService

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_UserService (uint8 Xcp_Channel, ROMBYTEPTR pCmd)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro XCP_CHANNEL_IDX to get the channel index.
pCmd	Pointer to XCP command packet
Return code	
uint8	XCP_CMD_OK : positive response XCP_CMD_PENDING : Call XcpSendCrm() when done XCP_CMD_SYNTAX : negative response
Functional Description	
Application specific user command. Please refer to 3.12 User Defined Command.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch XCP_ENABLE_USER_COMMAND has to be defined 	

6.5.12 XcpAppl_OpenCmdIf: XCP command extension interface

XcpAppl_OpenCmdIf

Prototype

Single Channel	
Single Receive Channel	uint8 XcpAppl_OpenCmdIf (uint8 Xcp_Channel, ROMBYTEPTR pCmd BYTEPTR pRes, BYTEPTR pLength)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro XCP_CHANNEL_IDX to get the channel index.
pCmd	Pointer to COMMAND that has been received by the XCP Slave.
pRes	Pointer to response buffer that will be sent by the XCP Slave.
pLength	Number of bytes that will be sent in the response.
Return code	
uint8	XCP_CMD_OK : Done XCP_CMD_PENDING : Call Xcp_SendCrm() when done XCP_CMD_ERROR : Error
Functional Description	
Call back that can be used to extend the XCP commands of the XCP protocol layer.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly > Call context: Task and interrupt level > The switch XCP_ENABLE_OPENCMDIF has to be defined 	

6.5.13 XcpAppl_SendStall: Resolve a transmit stall condition

XcpAppl_SendStall

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_SendStall (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
uint8	0 : if not successful > 0 : successful
Functional Description	
Resolve a transmit stall condition in Xcp_Putchar or Xcp_SendEvent.	

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > The switch `XCP_ENABLE_SEND_EVENT` or `XCP_ENABLE_SERV_TEXT_PUTCHAR` and `XCP_ENABLE_SEND_QUEUE` are defined
- > The function can be overwritten by the macro `XcpAppl_SendStall()`

6.5.14 XcpAppl_DisableNormalOperation: Disable normal operation of the ECU

XcpAppl_DisableNormalOperation

Prototype

Single Channel

Single Receive Channel	uint8 XcpAppl_DisableNormalOperation (MTABYTEPTR a, uint16 size)
------------------------	---------------------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

a	Address (where the flash kernel is downloaded to)
size	Size (of the flash kernel)

Return code

uint8	<code>XCP_CMD_OK</code> : download of flash kernel confirmed <code>XCP_CMD_DENIED</code> : 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

- > XCP is initialized correctly and in connected state
- > The switch `XCP_ENABLE_BOOTLOADER_DOWNLOAD` has to be defined

6.5.15 XcpAppl_StartBootLoader: Start of boot loader

XcpAppl_StartBootLoader

Prototype

Single Channel

Single Receive Channel	uint8 XcpAppl_StartBootLoader (void)
------------------------	-----------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

-	-
---	---

Return code	
uint8	<p>This function should not return.</p> <p>XCP_CMD_OK : positive response</p> <p>XCP_CMD_BUSY : negative response</p>
Functional Description	
Start of the boot loader.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch XCP_ENABLE_BOOTLOADER_DOWNLAOD has to be defined 	

6.5.16 XcpAppl_Reset: Perform ECU reset

XcpAppl_Reset

Prototype	
Single Channel	
Single Receive Channel	void XcpAppl_Reset (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Perform an ECU reset after reprogramming of the application.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch XCP_ENABLE_PROGRAM has to be defined 	

6.5.17 XcpAppl_ProgramStart: Prepare flash programming

XcpAppl_ProgramStart

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_ProgramStart (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported

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	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch XCP_ENABLE_PROGRAM has to be defined 	

6.5.18 XcpAppl_FlashClear: Clear flash memory

XcpAppl_FlashClear

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_FlashClear (MTABYTEPTR address, uint32 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
address	Address
size	Size
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	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switch XCP_ENABLE_PROGRAM has to be defined 	

6.5.19 XcpAppl_FlashProgram: Program flash memory

XcpAppl_FlashProgram

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_FlashProgram (ROMBYTEPTR data, MTABYTEPTR address, uint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
data	Pointer to data
address	Address
size	Size
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	
> XCP is initialized correctly and in connected state > The switch XCP_ENABLE_PROGRAM has to be defined	

6.5.20 XcpAppl_DaqResume: Resume automatic data transfer

XcpAppl_DaqResume

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_DaqResume (uint8 Xcp_Channel, tXcpDaq * daq)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro XCP_CHANNEL_IDX to get the channel index.
daq	Pointer to dynamic DAQ list structure
Return code	
uint8	0 : No resume mode data available >0 : Resume mode initialization ok

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

- > XCP is initialized correctly and in connected state
- > The switches `XCP_ENABLE_DAQ` and `XCP_ENABLE_DAQ_RESUME` are defined

6.5.21 XcpAppl_DaqResumeStore: Store DAQ lists for resume mode

XcpAppl_DaqResumeStore

Prototype

Single Channel

Single Receive Channel	<code>void XcpAppl_DaqResumeStore (uint8 Xcp_Channel, P2CONST(tXcpDaq, AUTOMATIC, XCP_APPL_DATA) daq , uint16 size, uint8 measurementStart)</code>
------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

<code>Xcp_Channel</code>	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
<code>daq</code>	Pointer to dynamic DAQ list structure.
<code>size</code>	Size of DAQ data that needs to be stored
<code>MeasurementStart</code>	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

- > XCP is initialized correctly and in connected state
- > The switches `XCP_ENABLE_DAQ` and `XCP_ENABLE_DAQ_RESUME` are defined

6.5.22 XcpAppl_DaqResumeClear: Clear stored DAQ lists

XcpAppl_DaqResumeClear

Prototype

Single Channel

Single Receive Channel	void XcpAppl_DaqResumeClear (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
Return code	
-	-
Functional Description	
The whole dynamic DAQ list structure that had been stored in non-volatile memory within the service <code>XcpAppl_DaqResumeStore(. .)</code> has to be cleared.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_DAQ</code> and <code>XCP_ENABLE_DAQ_RESUME</code> are defined 	

6.5.23 XcpAppl_CalResumeStore: Store Calibration data for resume mode

XcpAppl_CalResumeStore

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_CalResumeStore (uint8 Xcp_Channel)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
Return code	
uint8	0 : Storing not yet finished (STORE_CAL_REQ flag kept) >0 : Storing finished (STORE_CAL_REQ flag cleared)
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	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_DAQ</code> and <code>XCP_ENABLE_DAQ_RESUME</code> are defined 	

6.5.24 XcpAppl_GetTimestamp: Returns the current timestamp

XcpAppl_GetTimestamp

Prototype	
Single Channel	
Single Receive Channel	XcpDaqTimestampType XcpAppl_GetTimestamp (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
XcpDaqTimestampType	timestamp
Functional Description	
Returns the current timestamp.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_DAQ</code> and <code>XCP_ENABLE_DAQ_TIMESTAMP</code> are defined > The parameter <code>kXcpDaqTimestampSize</code> defines the timestamp size. It can either be <code>DAQ_TIMESTAMP_BYTE</code>, <code>DAQ_TIMESTAMP_WORD</code>, <code>DAQ_TIMESTAMP_DWORD</code> 	

6.5.25 XcpAppl_GetCalPage: Get calibration page

XcpAppl_GetCalPage

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_GetCalPage (uint8 Xcp_Channel, uint8 segment, uint8 mode)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
segment	Logical data segment number
mode	Access mode The access mode can be one of the following values: <code>CAL_ECU</code> : ECU access <code>CAL_XCP</code> : 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

- > XCP is initialized correctly and in connected state
- > The switches `XCP_ENABLE_DAQ` and `XCP_ENABLE_DAQ_TIMESTAMP` are defined

6.5.26 XcpAppl_SetCalPage: Set calibration page

XcpAppl_SetCalPage

Prototype

Single Channel

Single Receive Channel	uint8 XcpAppl_SetCalPage (uint8 Xcp_Channel, uint8 segment, uint8 page, uint8 mode)
------------------------	----------------------------------------------------------------------------------------------

Multi Channel

Indexed	not supported
---------	---------------

Code replicated	not supported
-----------------	---------------

Parameter

Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
segment	Logical data segment number
Page	Logical data page number
mode	Access mode <code>CAL_ECU</code> : the given page will be used by the slave device application <code>CAL_XCP</code> : the slave device XCP driver will access the given page Both flags may be set simultaneously or separately.

Return code

uint8	<code>XCP_CMD_OK</code> : Operation completed successfully <code>XCP_CMD_PENDING</code> : Call <code>Xcp_SendCrm()</code> when done <code>CRC_OUT_OF_RANGE</code> : segment out of range (only one segment supported) <code>CRC_PAGE_NOT_VALID</code> : Selected page not available <code>CRC_PAGE_MODE_NOT_VALID</code> : Selected page mode not available
-------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Functional Description

Set the access mode for a calibration data segment.

Particularities and Limitations

- > XCP is initialized correctly and in connected state
- > The switches `XCP_ENABLE_DAQ` and `XCP_ENABLE_DAQ_TIMESTAMP` are defined

6.5.27 XcpAppl_CopyCalPage: Copying of calibration data pages

XcpAppl_CopyCalPage

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_CopyCalPage (uint8 Xcp_Channel, uint8 srcSeg, uint8 srcPage, uint8 destSeg, uint8 destPage)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
srcSeg	Source segment
srcPage	Source page
destSeg	Destination segment
destPage	Destination page
Return code	
uint8	<code>XCP_CMD_OK</code> : Operation completed successfully <code>XCP_CMD_PENDING</code> : Call <code>XcpSendCrm()</code> when done <code>CRC_PAGE_NOT_VALID</code> : Page not available <code>CRC_SEGMENT_NOT_VALID</code> : Segment not available <code>CRC_WRITE_PROTECTED</code> : Destination page is write protected.
Functional Description	
Copying of calibration data pages. The pages are copied from source to destination.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_PAGE_COPY</code> and <code>XCP_ENABLE_DAQ_TIMEOUT</code> are defined 	

6.5.28 XcpAppl_SetFreezeMode: Setting the freeze mode of a segment

XcpAppl_SetFreezeMode

Prototype	
Single Channel	
Single Receive Channel	void XcpAppl_SetFreezeMode (uint8 segment, uint8 mode)
Multi Channel	
Indexed	not supported
Code replicated	not supported
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	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_PAGE_FREEZE</code> is defined 	

6.5.29 XcpAppl_GetFreezeMode: Reading the freeze mode of a segment

XcpAppl_GetFreezeMode

Prototype	
Single Channel	
Single Receive Channel	uint8 XcpAppl_GetFreezeMode (uint8 segment)
Multi Channel	
Indexed	not supported
Code replicated	not supported
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	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_PAGE_FREEZE</code> is defined 	

6.5.30 XcpAppl_Read: Read a single byte from memory

XcpAppl_Read

Prototype	
Single Channel	
Single Channel	uint8 XcpAppl_Read (uint32 addr)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
addr	32 Bit address

Return code	
uint8	Pointer to the address specified by the parameters
Functional Description	
Read a single byte from the memory.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_MEM_ACCESS_BY_APPL</code> is defined 	

6.5.31 XcpAppl_Write: Write a single byte to RAM

XcpAppl_Write

Prototype	
Single Channel	
Single Channel	void XcpAppl_Write (uint32 addr, uint8 data)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
addr	32 Bit address
data	data to be written to memory
Return code	
-	-
Functional Description	
Write a single byte to RAM.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_MEM_ACCESS_BY_APPL</code> is defined 	

6.5.32 XcpAppl_MeasurementRead: Read multiple bytes from memory

XcpAppl_MeasurementRead

Prototype	
Single Channel	
Single Channel	uint8 XcpAppl_MeasurementRead (P2VAR(void, AUTOMATIC, XCP_APPL_DATA) dst, P2CONST(void, AUTOMATIC, XCP_APPL_DATA) src, uint8 len)
Multi Channel	
Indexed	not supported
Code replicated	not supported

Parameter	
dst	Address pointer
len	Number of bytes to read
src	Pointer to data
Return code	
uint8	XCP_CMD_OK if read operation was successful otherwise return protection code, e.g. XCP_CMD_DENIED
Functional Description	
Read multiple bytes from memory. This service is used in MultiCore use case for type safe read operation.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches XCP_ENABLE_CALIBRATION_MEM_ACCESS_BY_APPL or XCP_ENABLE_TYPESAVE_COPY is defined 	

6.5.33 XcpAppl_CalibrationWrite: Write multiple bytes to memory

XcpAppl_CalibrationWrite

Prototype	
Single Channel	
Single Channel	uint8 XcpAppl_CalibrationWrite (P2VAR(void, AUTOMATIC, XCP_APPL_DATA) dst, P2CONST(void, AUTOMATIC, XCP_APPL_DATA) src, uint8 len)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
dst	Address pointer
len	Number of bytes to write
src	Pointer to data
Return code	
uint8	Protection code, XCP_CMD_OK if write operation was successful
Functional Description	
Write multiple bytes to memory. This service is used in MultiCore use case for type safe write operation.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches XCP_ENABLE_CALIBRATION_MEM_ACCESS_BY_APPL or XCP_ENABLE_TYPESAVE_COPY is defined 	

6.5.34 XcpAppl_ReadChecksumValue: Read checksum value

XcpAppl_ReadChecksumValue

Prototype	
Single Channel	
Single Channel	tXcpChecksumAddType XcpAppl_ReadChecksumValue (uint32 addr)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Addr	Address pointer
Return code	
tXcpChecksumAddType	New value for checksum calculation
Functional Description	
This function is used to access checksum values when no direct access to memory is allowed.	
Particularities and Limitations	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_CALIBRATION_MEM_ACCESS_BY_APPL</code> is defined 	

6.5.35 XcpAppl_CalculateChecksum: Custom checksum calculation

XcpAppl_CalculateChecksum

Prototype	
Single Channel	
Single Channel	uint8 XcpAppl_CalculateChecksum (uint8 Xcp_Channel, ROMBYTEPTR pMemArea, BYTEPTR pRes, uint32 length)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
Xcp_Channel	A channel parameter, used when the multi client feature is active. Please use the macro <code>XCP_CHANNEL_IDX</code> to get the channel index.
pMemArea	Address pointer
pRes	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	
<ul style="list-style-type: none"> > XCP is initialized correctly and in connected state > The switches <code>XCP_ENABLE_CHECKSUM</code> and <code>XCP_ENABLE_CUSTOM_CRC</code> is defined 	

6.6 XCP Protocol Layer Functions that can be overwritten

The following functions are defined within the XCP Protocol Layer and can be overwritten for optimization purposes.

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

6.6.1 Xcp_MemCpy: Copying of a memory range

Xcp_MemCpy

Prototype	
Single Channel	
Single Receive Channel	void Xcp_MemCpy (DAQBYTEPTR dest, ROMDAQBYTEPTR src, uint8 n)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
dest	pointer to destination address
src	pointer to source address
n	number of data bytes to copy
Return code	
-	-

Functional Description

General memory copy function that copies a memory range from source to destination.

This function is used in the inner loop of `Xcp_Event` for data acquisition sampling.

This function is already defined in the XCP Protocol Layer, but can be overwritten by a macro or function for optimization purposes. E.g. it would be possible to use DMA for faster execution.

Particularities and Limitations

- > The XCP Protocol Layer has been initialized correctly.
- > This function can be overwritten if `Xcp_MemCpy` is defined.

6.6.2 Xcp_MemSet: Initialization of a memory range

Xcp_MemSet

Prototype

Single Channel

Single Receive Channel `void Xcp_MemSet (BYTEPTR p, uint16 n, uint8 b)`

Multi Channel

Indexed not supported

Code replicated not supported

Parameter

<code>p</code>	pointer to start address
<code>n</code>	number of data bytes
<code>b</code>	data byte to initialize with

Return code

-	-
---	---

Functional Description

Initialization of `n` bytes starting from address `p` with `b`.

This function is already defined in the XCP Protocol Layer, but can be overwritten by a macro or function for optimization purposes. E.g. it would be possible to use DMA for faster execution.

Particularities and Limitations

- > The XCP Protocol Layer has been initialized correctly.
- > This function can be overwritten if `Xcp_MemSet` is defined.

6.6.3 Xcp_MemClr: Clear a memory range

Xcp_MemClr

Prototype

Single Channel

Single Receive Channel `static void Xcp_MemClr (BYTEPTR p, uint16 n)`

Multi Channel

Indexed not supported

Code replicated not supported

Parameter	
p	pointer to start address
n	number of data bytes
Return code	
-	-
Functional Description	
Initialize n data bytes starting from address p with 0x00. This function is already defined in the XCP Protocol Layer, but can be overwritten by a macro or function for optimization purposes. E.g. it would be possible to use DMA for faster execution.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The XCP Protocol Layer has been initialized correctly. > This function can be overwritten if <code>Xcp_MemClr</code> is defined. 	

6.7 AUTOSAR CRC Module Services called by the XCP Protocol Layer (XCP Professional Only)

The following services of the AUTOSAR CRC Module are called by the XCP Protocol Layer:

`Crc_CalculateCRC16 (...)`

`Crc_CalculateCRC32 (...)`

A detailed description of the API can be found in the software specification of the CRC Module [VII].

6.7.1.1 Generated a2l files

The GenTool also generates multiple a2l files which can be used in the Master 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
  ...

```

	<pre>/end DAQ /include CanXCPAsr.a2l /end IF_DATA ... /include bsw.a2l ...</pre>
--	-----------------------------------------------------------------------------------------------------------------

6.8 Configuration without Generation Tool

The configuration of the configuration switches and constants is done in the file `Xcp_Cfg.h`.

6.8.1 Compiler Switches

Compiler switches are used to enable/disable optional functionalities in order to save code space and RAM.

In the following table you will find a complete list of all configuration switches, used to control the functional units. The default values are bold.

Configuration switches	Value	Description
<code>XCP_XXX_DAQ</code>	ENABLE, DISABLE	Enables/disables synchronous data acquisition.
<code>XCP_XXX_DAQ_PRESCALER</code>	ENABLE, DISABLE	Enables/disables the DAQ prescaler.
<code>XCP_XXX_DAQ_OVERRUN_INDICATION</code>	ENABLE, DISABLE	Enables/disables the DAQ overrun detection.
<code>XCP_XXX_DAQ_HDR_ODT_DAQ²</code>	ENABLE, DISABLE	The 2 Byte DAQ/ODT XCP Packet identification is used instead of the PID. Enabled: Relative ODT number, absolute list number (BYTE) Disabled: Absolute ODT number
<code>XCP_XXX_DAQ_PROCESSOR_INFO</code>	ENABLE, DISABLE	Plug & play mechanism for the data acquisition processor.
<code>XCP_XXX_DAQ_RESOLUTION_INFO</code>	ENABLE, DISABLE	Plug & play mechanism for the data acquisition resolution.
<code>XCP_XXX_DAQ_EVENT_INFO</code>	ENABLE, DISABLE	Plug & play mechanism for the event definitions.
<code>XCP_XXX_DAQ_TIMESTAMP</code>	ENABLE, DISABLE	DAQ timestamps

² The XCP Protocol allows three identification field types for DTOs: 'absolute ODT number', 'relative ODT number and absolute DAQ list number', 'empty identification field' (not supported)

XCP_XXX_DAQ_TIMESTAMP_FIXED	ENABLE, DISABLE	Slave always sends DTO Packets in time stamped mode. Otherwise are timestamps used individual by each DAQ-list.
kXcpDaqTimestampSize	DAQ_TIMESTAMP_BYTE, DAQ_TIMESTAMP_WORD, DAQ_TIMESTAMP_DWORD	The size of timestamps which can either be 1Byte, 2Bytes or 4Bytes.
XCP_XXX_SEED_KEY	ENABLE, DISABLE	Seed & key access protection
XCP_XXX_CHECKSUM	ENABLE, DISABLE	Calculation of checksum
XCP_XXX_CUSTOM_CRC	ENABLE, DISABLE	Enable call-back for custom CRC calculation
XCP_XXX_CRC16CCITT_REFLECTED	ENABLE, DISABLE	Enable/disable reflected CRC16 CCITT checksum calculation algorithm. Also refer to 6.8.2.1 'Table of Checksum Calculation Methods'.
XCP_XXX_AUTOSAR_CRC_MODULE	ENABLE , DISABLE	Usage of CRC algorithms of AUTOSAR CRC module.
XCP_XXX_PARAMETER_CHECK	ENABLE, DISABLE	Parameter check
XCP_XXX_SEND_QUEUE	ENABLE , DISABLE	Transmission send queue (shall be used in conjunction with synchronous data acquisition and stimulation).
XCP_XXX_SEND_EVENT	ENABLE, DISABLE	Transmission of event packets (EV)
XCP_XXX_USER_COMMAND	ENABLE, DISABLE	User defined command
XCP_XXX_GET_ID_GENERIC	ENABLE, DISABLE	ECU identification
XCP_XXX_TL_COMMAND	ENABLE, DISABLE	Transport Layer command
XCP_XXX_COMM_MODE_INFO	ENABLE, DISABLE	Communication mode info
XCP_XXX_CALIBRATION_PAGE	ENABLE, DISABLE	Calibration data page switching

XCP_XXX_PAGE_INFO	ENABLE, DISABLE	Calibration data page plug & play mechanism
XCP_XXX_PAGE_COPY	ENABLE, DISABLE	Calibration data page copying
XCP_XXX_PAGE_FREEZE	ENABLE, DISABLE	Segment freeze mode handling
XCP_XXX_DPRAM ³	ENABLE, DISABLE	Supports the usage of dual port RAM
XCP_XXX_BLOCK_UPLOAD	ENABLE, DISABLE	Enables/disables the slave block transfer.
XCP_XXX_BLOCK_DOWNLOAD	ENABLE, DISABLE	Enables/disables the master block transfer.
XCP_XXX_WRITE_PROTECTION	ENABLE, DISABLE	Write access to RAM
XCP_XXX_READ_PROTECTION	ENABLE, DISABLE	Read access to RAM
XCP_XXX_READ_EEPROM	ENABLE, DISABLE	Read access to EEPROM
XCP_XXX_WRITE_EEPROM	ENABLE, DISABLE	Write access to EEPROM
XCP_XXX_PROGRAMMING_WRITE_PROTECTION	ENABLE, DISABLE	Write access to flash
XCP_XXX_PROGRAM	ENABLE, DISABLE	Flash programming
XCP_XXX_PROGRAM_INFO	ENABLE, DISABLE	Flash programming plug & play mechanism
XCP_XXX_BOOTLOADER_DOWNLOAD	ENABLE, DISABLE	Flash programming with a flash kernel
XCP_XXX_STIM	ENABLE, DISABLE	Enables/disables data stimulation. (also XCP_ENABLE_DAQ has to be defined in order to use data stimulation)
XCP_XXX_DAQ_RESUME	ENABLE, DISABLE	Data acquisition resume mode.
XCP_XXX_SERV_TEXT	ENABLE, DISABLE	Transmission of service request codes
XCP_XXX_SERV_TEXT_PUTCHAR	ENABLE, DISABLE	Putchar function for the transmission of service request messages
XCP_XXX_SERV_TEXT_PRINTF	ENABLE, DISABLE	Print function for the transmission of service request messages

³ Not supported by XCP Professional

XCP_XXX_MEM_ACCESS_BY_APPL	ENABLE, DISABLE	Memory access by application
XCP_XXX_MODEL_PAGED	ENABLE, DISABLE	Support for paging / banking
XCP_XXX_SHORT_DOWNLOAD	ENABLE, DISABLE	Support for SHORT_DOWNLOAD command
XCP_XXX_MODIFY_BITS	ENABLE, DISABLE	Support for MODIFY_BITS command
XCP_XXX_WRITE_DAQ_MULTIPLE	ENABLE, DISABLE	Write DAQ multiple command
XCP_XXX_GET_XCP_DATA_POINTER	ENABLE, DISABLE	Enable API for internal data access
XCP_XXX_CONTROL	ENABLE, DISABLE	Enable functionality to en- / disable XCP module
XCP_XXX_DEV_ERROR_DETECT	ENABLE, DISABLE	Enable Development Error check
XCP_XXX_READCCCONFIG	ENABLE, DISABLE	Enable Read of FlexRay Parameters
XCP_ADDR_EXT_READCCCONFIG	0x00...0xff	Address Extension to be used for FlexRay Parameters
XCP_XXX_VECTOR_GENERICMEASUREMENT	ENABLE, DISABLE	Support for Generic Measurement feature
XCP_XXX_GET_SESSION_STATUS_API	ENABLE, DISABLE	Enable API to acquire the current session status

6.8.2 Configuration of Constant Definitions

The configuration of constant definitions is done as described below.
The default values are bold.

Constant definitions	Range	Default	Description
kXcpMaxCTOMax	8..255	8	Maximum length of XCP command transfer objects (CTO). The length of the CTO can be variable. However it has to be configured according to the used XCP Transport Layer.

kXcpMaxDTOMax	8..255 ⁴	8	Maximum length of XCP data transfer objects (DTO). The length of the DTO can be variable. However it has to be configured according to the used XCP Transport Layer.
kXcpDaqMemSize	0..0xFFFF	256	Define the amount of memory used for the DAQ lists and buffers. Also refer to chapter 7 (Resource Requirements).
kXcpSendQueueMinSize	1..0x7F	-	The minimum queue size required for DAQ. The queue size is the unallocated memory reserved by kXcpDaqMemSize.
kXcpMaxEvent	0..0xFF ⁵	-	Number of available events in the slave (part of event channel plug & play mechanism) Also refer to chapter 6.8.5.
kXcpStimOdtCount	0..0xC0	0xC0	Maximum number of ODTs that may be used for Synchronous Data Stimulation.
kXcpChecksumMethod	-	-	Checksum calculation method. Refer to chapter 6.8.2.1 'Table of Checksum Calculation Methods' for valid values.
kXcpChecksumBlockSize	1..0xFFFF	256	Each call of Xcp_MainFunction calculates the checksum on the amount of bytes specified by kXcpChecksumBlockSize.
XCP_TRANSPORT_LAYER_VERSION	0..0xFFFF	-	Version of the XCP Transport Layer that is used. (this version gets transferred to the MCS)
kXcpMaxSector	1..0xFF	-	Number of flash sectors Also refer to chapter 6.8.7
kXcpMaxSegment	1	1	Number of memory segments Also refer to chapter 6.8.8.
kXcpMaxPages	1..2	2	Number of pages Also refer to chapter 6.8.8.
NUMBER_OF_TRANSPORTLAYERS	1..	1	Number of used Transport Layers
XCP_TRANSPORT_LAYER_CAN	0..	0	Index of Transport Layer
XCP_TRANSPORT_LAYER_FIR	0..	1	Index of Transport Layer
XCP_TRANSPORT_LAYER_ETH	0..	2	Index of Transport Layer

6.8.2.1 Table of Checksum Calculation Methods

Constant	Checksum calculation method
XCP_CHECKSUM_TYPE_ADD11	Add BYTE into a BYTE checksum, ignore overflows.
XCP_CHECKSUM_TYPE_ADD12	Add BYTE into a WORD checksum, ignore overflows

⁴ Implementation specific range. The range is 8..0xFFFF according to XCP specification [I], [II].

⁵ Implementation specific range. The range is 0..0xFFFE according to XCP specification [I], [II].

XCP_CHECKSUM_TYPE_ADD14	Add BYTE into a DWORD checksum, ignore overflows
XCP_CHECKSUM_TYPE_ADD22	Add WORD into a WORD checksum, ignore overflows, block size must be modulo 2
XCP_CHECKSUM_TYPE_ADD24	Add WORD into a DWORD checksum, ignore overflows, block size must be modulo 2
XCP_CHECKSUM_TYPE_ADD44	Add DWORD into DWORD, ignore overflows, block size must be modulo 4
XCP_CHECKSUM_TYPE_CRC16CCITT	CRC16 CCITT checksum calculation algorithm Both the standard and the reflected algorithm are supported. Please refer to chapter 9.6 'Reflected CRC16 CCITT Checksum Calculation Algorithm'. The CRC16 CCITT algorithm of the AUTOSAR CRC module is only supported by XCP Professional.
XCP_CHECKSUM_TYPE_CRC32	CRC32 checksum calculation algorithm The CRC32 algorithm is only supported in XCP Professional if the AUTOSAR CRC module is used.

6.8.3 Configuration of the CPU Type

To provide platform independent code platform, the CPU type has to be defined.

Configuration switches	Value	Description
C_CPUYPE_xxxENDIAN	LITTLE, BIG	Definition whether the CPU is little endian (Intel format) or big endian (Motorola format).
XCP_xxx_UNALIGNED_MEM_ACCESS	ENABLE, DISABLE	Enables / disables unaligned memory access. If XCP_DISABLE_UNALIGNED_MEM_ACCESS is defined WORDs are located on WORD aligned and DWORD are located on DWORD aligned addresses.

6.8.4 Configuration of Slave Device Identification

The configuration of the slave device identification and automatic session configuration is described within this chapter. Only one of the following options can be used at one time.

6.8.4.1 Identification by ASAM-MC2 Filename without Path and Extension

If the slave device identification is done by identification with an ASAM-MC2 filename without path and extension the filename length has to be defined:

```
#define kXcpStationIdLength length
```

and the station ID itself has to be defined as string:

```
const uint8 kXcpStationId[] = "station ID"
```

The range of kXcpStationIdLength is 0..0xFF.

6.8.4.2 Automatic Session Configuration with MAP Filenames

The automatic session configuration by transferring MAP filenames is a Vector specific extension that works with CANape and can be enabled by the "XcpGetIdGeneric" attribute.

When this feature is enabled the API as described in 3.4.2 XCP Generic Identification is enabled. This API will be called, should CANape request the MAP filename, and must be

implemented by the user accordingly. This feature must explicitly be enabled in CANape as well!



Example

```
#define MAP_FORMAT 29
#define MAP_NAME "xcpsim"

uint8 MapTest[500];
uint32 MapTestSize;

uint32 XcpAppl_GetIdData( MTABYTEPTR *pData, uint8 id )
{
    if( id == IDT_VECTOR_MAPNAMES )
    {
        MapTestSize =
        sprintf((char*)MapTest, "%c%c%s.map", MAP_FORMAT, 0, MAP_NAME);
        /* Result: MapTest = "290xcpsim.map" */
        *pData = MapTest;
        return MapTestSize;
    }
    else
    {
        return 0; /* Id not available */
    }
}
```

'MAP_FORMAT' represents the format of the MAP file. (See table below)

'0' is a counter that is used as address extension. Please set this parameter to 0.

Table of MAP file formats:

1 = "BorlandC 16 Bit"	29 = "Microsoft standard"
2 = "M166"	30 = "ELF/DWARF 16 Bit"
3 = "Watcom"	31 = "ELF/DWARF 32 Bit"
4 = "HiTech HC05"	32 = "Fujitsu Softune 3..8(.mps)"
6 = "IEEE"	33 = "Microware Hawk"
7 = "Cosmic"	34 = "TI C6711"
8 = "SDS"	35 = "Hitachi H8S"
9 = "Fujitsu Softune 1(.mp1)"	36 = "IAR HC12"
10 = "GNU"	37 = "Greenhill Multi 2000"
11 = "Keil 16x"	38 = "LN308(MITSUBISHI) for M16C/80"
12 = "BorlandC 32 Bit"	39 = "COFF settings auto detected"
13 = "Keil 16x (static)"	40 = "NEC CC78K/0 v35"
14 = "Keil 8051"	41 = "Microsoft extended"
15 = "ISI"	42 = "ICCAVR"
16 = "Hiware HC12"	43 = "Omf96 (.m96)"
17 = "TI TMS470"	44 = "COFF/DWARF"

18 = "Archimedes"	45 = "OMF96 Binary (Tasking C196)"
19 = "COFF"	46 = "OMF166 Binary (Keil C166)"
20 = "IAR"	47 = "Microware Hawk Plug&Play ASCII"
21 = "VisualDSP"	48 = "UBROF Binary (IAR)"
22 = "GNU 16x"	49 = "Renesas M32R/M32192 ASCII"
23 = "GNU VxWorks"	50 = "OMF251 Binary (Keil C251)"
24 = "GNU 68k"	51 = "Microsoft standard VC8"
25 = "DiabData"	52 = "Microsoft VC8 Release Build (MATLAB DLL)"
26 = "VisualDSP DOS"	53 = "Microsoft VC8 Debug Build (MATLAB DLL)"
27 = "HEW SH7055"	54 = "Microsoft VC8 Debug file (pdb)"
28 = "Metrowerks"	

6.8.5 Configuration of the Event Channel Plug & Play Mechanism

The event channel plug & play mechanism is enabled with the switch

```
XCP_ENABLE_DAQ_EVENT_INFO
```

A prerequisite for the event channel plug & play mechanism is the general data acquisition plug & play mechanism. If the mechanism is enabled the following configurations items have to be defined as described below:

Constant	Range	Description
kXcpMaxEvent	0..0xFF ⁶	Number of available events in the slave (part of event channel plug & play mechanism) If the event numbers do not start at 0 or are not continuous this is the maximum used event channel number plus 1.
kXcpEventName[]	kXcpMaxEvent	List with pointers to the event channel names that are defined as strings.
kXcpEventNameLength[]	kXcpMaxEvent	Length of the event channel names without the terminating char.
kXcpEventCycle[]	kXcpMaxEvent	Cycle time of the event channels in milliseconds.
kXcpEventDirection[]	kXcpMaxEvent	Direction of the event channels. For XCP Basic valid values are: <ul style="list-style-type: none"> - kXcpEventDirectionDaq For XCP Professional valid values are: <ul style="list-style-type: none"> - kXcpEventDirectionDaq - kXcpEventDirectionStim - kXcpEventDirectionDaqStim



Example

```
#define XCP_ENABLE_DAQ_EVENT_INFO
#define kXcpMaxEvent 3

CONST(uint8, XCP_CONST) kXcpEventName_0[] = "10ms";
```

⁶ Implementation specific range. The range is 0..0xFFFE according to XCP specification [I], [II].

```

CONST(uint8, XCP_CONST) kXcpEventName_1[] = "100ms DAQ";
CONST(uint8, XCP_CONST) kXcpEventName_2[] = "100ms STIM";
CONSTP2CONST(uint8, XCP_CONST, XCP_CONST) kXcpEventName[] =
{
    &kXcpEventName_0[0],
    &kXcpEventName_1[0],
    &kXcpEventName_2[0]
};

CONST(uint8, XCP_CONST) kXcpEventNameLength[] =
{
    4,
    9,
    10
};

CONST(uint8, XCP_CONST) kXcpEventCycle[] =
{
    10,
    100,
    100
};

CONST(uint8, XCP_CONST) kXcpEventDirection[] =
{
    kXcpEventDirectionDaq,
    kXcpEventDirectionDaq,
    kXcpEventDirectionStim
};

```

6.8.6 Configuration of the DAQ Time Stamped Mode

Transmission of DAQ timestamps is enabled with `XCP_ENABLE_DAQ_TIMESTAMP`. If `XCP_ENABLE_DAQ_TIMESTAMP_FIXED` is defined all DTO Packets will be transmitted in time stamped mode.

Constant	Range	Description
<code>kXcpDaqTimestampSize</code>	<code>DAQ_TIMESTAMP_BYTE</code> , <code>DAQ_TIMESTAMP_WORD</code> , <code>DAQ_TIMESTAMP_DWORD</code>	This parameter defines the size of timestamps. It can either be 1 byte, 2 bytes or 4 bytes.
<code>XcpDaqTimestampType</code>	<code>uint8</code> , <code>uint16</code> or <code>uint32</code>	Type of the timestamp depends on the parameter <code>kXcpDaqTimestampSize</code> .

kXcpDaqTimestampUnit	DAQ_TIMESTAMP_UNIT_1NS	Unit of the timestamp (1 ns, 10 ns .. 1 s)
	DAQ_TIMESTAMP_UNIT_10NS	
	DAQ_TIMESTAMP_UNIT_100NS	
	DAQ_TIMESTAMP_UNIT_1US	
	DAQ_TIMESTAMP_UNIT_10US	
	DAQ_TIMESTAMP_UNIT_100US	
	DAQ_TIMESTAMP_UNIT_1MS	
	DAQ_TIMESTAMP_UNIT_10MS	
	DAQ_TIMESTAMP_UNIT_100MS	
	DAQ_TIMESTAMP_UNIT_1S	
	DAQ_TIMESTAMP_UNIT_1pS	
	DAQ_TIMESTAMP_UNIT_10pS	
	DAQ_TIMESTAMP_UNIT_100pS	
kXcpDaqTimestampTicksPerUnit	0..0xFFFF	Time stamp ticks per unit

6.8.7 Configuration of the Flash Programming Plug & Play Mechanism

The flash programming plug & play mechanism is enabled with the switch

```
XCP_ENABLE_PROGRAM_INFO
```

If the plug & play mechanism is enabled the number of sectors and the start address and end address of each sector has to be defined. The constants that have to be defined can be found in the following table.

Constant	Range	Description
kXcpMaxSector	0..0xFF	Number of available flash sectors in the slave
kXcpSectorName[]	kXcpMaxSector	List with pointers to the Sector names that are defined as strings.
kXcpSectorNameLength	kXcpMaxSector	Length of the Sector names without the terminating char.
kXcpProgramSectorStart[]	kXcpMaxSector	List with the start addresses of the sectors
kXcpProgramSectorEnd[]	kXcpMaxSector	List with the end address of the sectors



Example

```
#define XCP_ENABLE_PROGRAM_INFO
#define kXcpMaxSector 2

CONST(XcpCharType, XCP_CONST) kXcpSectorName_0[] = "Sector0";
CONST(XcpCharType, XCP_CONST) kXcpSectorName_1[] = "Sector1";

CONSTP2CONST(XcpCharType, XCP_CONST, XCP_CONST) kXcpSectorName[] =
{
    &kXcpSectorName_0[0],
    &kXcpSectorName_1[0]
};
CONST(uint8, XCP_CONST) kXcpSectorNameLength[] =
{
```

```

7U,
7U
};
CONST(uint32, XCP_CONST) kXcpProgramSectorStart [] =
{
    (uint32)0x000000u,
    (uint32)0x010000u,
};
CONST(uint32, XCP_CONST) kXcpProgramSectorEnd [] =
{
    (uint32)0x00FFFFu,
    (uint32)0x01FFFFu,
};

```

6.8.8 Configuration of the Page Switching Plug & Play Mechanism

The page switching plug & play mechanism is enabled with the switch

`XCP_ENABLE_PAGE_INFO`

If the plug & play mechanism is enabled the following configurations items have to be defined as described below:

Constant	Range	Description
<code>kXcpMaxSegment</code>	<code>0x01</code>	Number of memory segments
<code>kXcpMaxPages</code>	<code>0x01..0x02</code>	Number of pages

6.8.9 Configuration of the used Transport Layer

The XCP Protocol Layer uses a jump table to call respective Transport Layer Functions. This jump table has to contain certain Function names

Constant	Range	Description
<code>Xcp_TlApi</code>	Number of TL	Function Pointer table containing pointers to the respective Transport Layer



Example

```

#define NUMBER_OF_TRANSPORTLAYERS 1
#define XCP_TRANSPORT_LAYER_CAN 0u

CONST(Xcp_TIApiType, XCP_CONST)
Xcp_TIApi[NUMBER_OF_TRANSPORTLAYERS] =
{
    {
        CanXcp_Send, /* ApplXcpSend */
        CanXcp_SendFlush /* ApplXcpSendFlush */
        #if defined ( XCP_ENABLE_TL_COMMAND )
        ,
        CanXcp_TLService /* ApplXcpTLService */
        #endif
    }
};

```

--	--

7 Resource Requirements

The resource requirements of the XCP Protocol Layer mainly depend on the micro controller, compiler options and configuration. Within this chapter only the configuration specific resource requirements are taken in consideration.

8 Limitations

8.1 General Limitations

The functional limitations of the XCP Professional Version are listed below:

- > Bit stimulation is not supported
- > Only dynamic DAQ list allocation supported
- > The interleaved communication model is not supported
- > Only default programming data format is supported
- > GET_SECTOR_INFO does not return sequence numbers
- > Program Verify and Program Format are not supported
- > DAQ numbers are limited to byte size
- > DAQ does not support address extension
- > DAQ-list and event channel prioritization is not supported
- > Event channels contain one DAQ-list
- > ODT optimization not supported
- > Assignments of CAN identifiers to DAQ lists is not supported
- > MAX_DTO is limited to 0xFF
- > The resume bits in DAQ lists are not set
- > STORE_DAQ, CLEAR_DAQ and STORE_CAL do not send an event message
- > Entering resume mode does not send an event message
- > Overload indication by an event is not supported
- > SERV_RESET is not supported
- > The following checksum types are not supported
 - > XCP_CRC_16
 - > XCP_CRC_32
 - > XCP_USER_DEFINED
- > Maximum checksum block size is 0xFFFF
- > Page Info and Segment Info is not supported
- > Only one segment and two pages are supported
- > The seed size and key size must be equal or less MAX_CTO-2
- > Consistency only supported on ODT level

Planned:

- > User defined checksum calculations
- > CRC16 and CRC32
- > The AUTOSAR API Xcp_SetTransmissionMode is not supported

8.2 Limitations Regarding Platforms, Compilers and Memory Models

Even though the XCP is a Protocol Layer and therefore higher software layer, it manipulates memory addresses and directly access the memory with these addresses.

This might cause issues for some combinations of platforms, compilers and memory models. The following list provides all known restrictions on platforms, compilers and linkers:

- > CANoeOSEK Emulation is not supported

9 FAQ

9.1 Invalid Time Stamp Unit



FAQ

If using data acquisition CANape reports an error due to an invalid timestamp unit.

If you are using CANape 5.5.x or an earlier version please define

```
#define XCP_ENABLE_CANAPE_5_5_X_SUPPORT
```

in your user config file.

9.2 Support of small and medium memory model



FAQ

How is the XCP Protocol Layer configured in order to access the whole memory in the small and medium memory model?

By default The XCP Protocol Layer accesses the memory with a default pointer. I.e. in small and medium memory model a near pointer is used. If the far memory (e.g. code or read-only sections) needs to be accessed via the XCP Protocol the memory qualifiers have to be defined as far pointers by the user within the user config file.

Two memory qualifiers are used to access the memory:

MTABYTEPTR

```
#define MTABYTEPTR P2VAR(uint8, AUTOMATIC, XCP_MTA_DATA)
This pointer is used to access memory for standard read and
write operations
```

DAQBYTEPTR

```
#define DAQBYTEPTR P2VAR(uint8, AUTOMATIC, XCP_DAQ_DATA)
This pointer is used to access memory for the Synchronous Data
Acquisition
```

Depending on the use case, microcontroller, memory model and compiler either `XCP_MEMORY_FAR` or both memory qualifiers (`DAQBYTEPTR` and `MTABYTEPTR`) have to be defined by the user. Alternatively the AUTOSAR Compiler Abstraction can be used. In this case the pointer classes

`XCP_MTA_DATA` and

`XCP_DAQ_DATA`

Have to be defined as “far” according to the used compiler.

9.3 Small memory model on ST10 / XC16X / C16X with Tasking Compiler



FAQ

How has XCP Protocol Layer to be configured in order to support small memory model on the following microcontrollers: ST10, XC16X, C16X with Tasking Compiler?

If the small memory model is used and the two least significant bits of the DPP register where the data of XCP is located is not equal the default DPP register value (i.e. the two least significant bits of DPPx are unequal x, x=0..3) the configuration of the XCP Protocol Layer has to be adapted in the user config file

Disable type casts from pointers to integers :

```
#define XCP_ENABLE_NO_P2INT_CAST
```

9.4 Data Page Banking on Star12X / Metrowerks



FAQ

How has the XCP Protocol Layer to be configured in order to support data page banking on the Star12X with Metrowerks compiler?

In order to use data page banking the following definition has to be added to the user config file:

```
#define XCP_MEMORY_MODEL_PAGED
```

If this option is enabled far pointers are used for memory access, and address conversions are carried out in the in the application callback template `_xcp_appl.c`. These address conversions have to adapted to the used derivative.



Please note

The data page banking support is implemented in the template `_xcp_appl.c` for the MC9S12XDP512. For other Star12X derivatives the template has to be adapted.

9.5 Memory model banked on Star12X / Cosmic



FAQ

How has the XCP Protocol Layer to be configured in order to support the access to far pages in the banked memory model on the Star12X with Cosmic compiler?

In order to access far pages or support data page banking the following definitions have to be added to the user config file:

```
#define XCP_MEMORY_MODEL_PAGED
```

```
#define XCP_ENABLE_MEM_ACCESS_BY_APPL
```

If this option is enabled far pointers are used for memory access, and address conversions are carried out in the application callback template `_xcp_appl.c`. These address conversions have to be adapted to the used derivative.

**Please note**

The data page banking support is implemented in the template `_xcp_appl.c` for the MC9S12XDP512. For other Star12X derivatives the template has to be adapted.

9.6 Reflected CRC16 CCITT Checksum Calculation Algorithm

**FAQ**

How is the reflected CRC16 CCITT checksum calculation algorithm configured?

The XCP Protocol Layer supports both the standard CRC16 CCITT algorithm and the reflected CRC16 CCITT algorithm. In order to use the reflected algorithm the following definition has to be added to the user config file:

```
#define XCP_ENABLE_CRC16CCITT_REFLECTED
```

**Please note**

Up to CANape version 5.6.30.3 (SP3) the standard CRC16 CCITT algorithm is not supported, but the reflected one. However a user checksum calculation DLL can be used in order to use the standard algorithm with former versions of CANape.

10 Bibliography

This manual refers to the following documents:

- [I] XCP -Part 1 - Overview
Version 1.1
- [II] XCP -Part 2- Protocol Layer Specification
Version 1.1
- [III] XCP -Part 5- Example Communication Sequences
Version 1.1
- [IV] Technical Reference XCP on CAN Transport Layer
Version 1.6
- [V] Technical Reference XCP on FlexRay Transport Layer
Version 1.9
- [VI] Technical Reference XCP on LIN Transport Layer
Version 1.0
- [VII] AUTOSAR Specification of CRC Routines
Release 2.0.0 of 2006-04-28

11 Contact

Visit our website for more information on

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

www.vector-informatik.com