XCP Protocol Layer

Technical Reference

Version 1.17.01

Authors:	Frank Triem, Sven Hesselmann, Andreas Herkommer
Version:	1.17.01
Status:	released

1 History

Author	Date	Version	Remarks
Frank Triem Klaus Emmert	2005-01-17	1.00.00	ESCAN00009143: Initial draft Warning Text added
Frank Triem	2005-06-22	1.01.00	FAQ extended: ESCAN00012356, ESCAN00012314 ESCAN00012617: Add service to retrieve XCP state
Frank Triem	2005-12-20	1.02.00	ESCAN00013883: Revise Resume Mode
Frank Triem	2006-03-09	1.03.00	ESCAN00015608: Support command TRANSPORT_LAYER_CMD ESCAN00015609: Support XCP on FlexRay Transport Layer
Frank Triem	2006-04-24	1.04.00	ESCAN00015913: Correct filenames Data page banking support of application callback template added
Frank Triem	2006-05-08	1.05.00	ESCAN00016263: Describe support of reflected CRC16 CCITT ESCAN00016159: Add demo disclaimer to XCP Basic
Frank Triem	2006-05-29	1.06.00	ESCAN00016226: Support XCP on LIN Transport Layer
Frank Triem	2006-07-20	1.07.00	ESCAN00012636: Add configuration with GENy ESCAN00016956: Support AUTOSAR CRC module
Frank Triem	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
Frank Triem	2007-01-18	1.09.00	ESCAN00018809: Support data paging on Star12X / Cosmic
Sven Hesselmann	2007-05-07	1.10.00	Description of new features added
Sven Hesselmann	2007-09-14	1.11.00	Segment freeze mode now supported
Andreas Herkommer	2008-07-23	1.12.00	ESCAN00028586: Support of Program_Start callback ESCAN00017955: Support MIN_ST_PGM ESCAN00017952: Open Interface for command processing
Sven Hesselmann	2008-09-10	1.13.00	Additional pending return value of call backs added MIN_ST configuration added
Andreas Herkommer	2008-12-01	1.14.00	ESCAN00018157: SERV_RESET is not supported ESCAN00032344: Update of XCP Basic Limitations
Andreas Herkommer	2009-05-14	1.15.00	ESCAN00033909: New features implemented: Prog Write Protection, Timestamps, Calibration activation

Klaus Bergdolt	2009-07-30	1.15.01	Fixed some editorial errors
Andreas Herkommer	2009-11-17	1.15.02	ESCAN00037907: XCP Memory in far RAM
Mario Kunz	2009-12-17	1.16.00	Support of a2l export
Andreas Herkommer	2012-02-20	1.16.01	ESCAN00055216: DAQ Lists can be extended after START_STOP_SYNCH
Andreas Herkommer	2012-08-13	1.17.00	ESCAN00060779: Support for address doubling in XCP for DSP micros
Andreas Herkommer	2013-06-18	1.17.01	ESCAN00068051: Provide an API to detect XCP state and usage

Contents

Illustrations

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 **A**SAM **2**MC **L**anguage File
AML	**A**SAM 2 **M**eta **L**anguage
API	**A**pplication **P**rogramming **I**nterface
ASAM	**A**ssociation for **S**tandardization of **A**utomation and **M**easuring Systems
BYP	**BYP**assing
CAN	**C**ontroller **A**rea **N**etwork
CAL	**CAL**ibration
CANape	Calibration and Measurement Data Acquisition for Electronic Control Systems

CMD	**C**o**m**man**d**
CTO	**C**ommand **T**ransfer **O**bject
DAQ	Synchronous **D**ata **A**c**q**uistion
DLC	**D**ata **L**ength **C**ode (Number of data bytes of a CAN message)
DLL	**D**ata **l**ink **l**ayer
DTO	**D**ata **T**ransfer **O**bject
ECU	**E**lectronic **C**ontrol **U**nit
ERR	**Err**or Packet
EV	**Ev**ent packet
ID	**Id**entifier (of a CAN message)
Identifier	Identifies a CAN message
ISR	**I**nterrupt **S**ervice **R**outine
MCS	**M**aster **C**alibration **S**ystem
Message	One or more signals are assigned to each message.
ODT	**O**bject **D**escriptor **T**able
OEM	**O**riginal **e**quipment **m**anufacturer (vehicle manufacturer)
PAG	**PAG**ing
PID	**P**acket **Id**entifier
PGM	**P**ro**g**ra**m**ming
RAM	**R**andom **A**ccess **M**emory
RES	Command **Res**ponse Packet
ROM	**R**ead **O**nly **M**emory
SERV	**Ser**vice Request Packet
STIM	**Stim**ulation
TCP/IP	**T**ransfer **C**ontrol **P**rotocol / **I**nternet **P**rotocol
UDP/IP	**U**nified **D**ata **P**rotocol / **I**nternet **P**rotocol
USB	**U**niversal **S**erial **B**us
XCP	Universal Measurement and **C**alibration **P**rotocol
VI	**V**ector **I**nformatik GmbH

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	
`Xcp…`	It is mandatory to use all functions beginning with Xcp… 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.
`ApplXcp…`	The functions, starting with `ApplXcp…` are functions that are provided either by any XCP Transport Layer or the application and are called by the XCP Protocol Layer. These services are user callback functions that are application specific and have to be implemented depending on the application.

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).

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 can the slave respond subsequent (there is no limitation) to a request without any more requests in between.

Refer to chapter 7.2.1 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 7.2.5.1 (Identification by ASAM-MC2 Filename without Path and Extension) for configuration details.

3.4.2 Transferring of XCP MAP Filenames

The MCS can also perform an automatic session configuration by transferring MAP filenames from the XCP slave to the master.

In this case the function

`vuint32 **ApplXcpGetIdData**( MTABYTEPTR *pData )` (6.5.2)

has to returns a pointer to a pointer of MAP file names.
Refer to chapter 7.2.5.2 (Automatic Session Configuration with MAP Filenames) for configuration details.

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 `XCP_DISABLE_SEED_KEY` in order to save ROM. Also refer to chapter 7.2.1.

The application callback function

`vuint8 **ApplXcpGetSeed**( MEMORY_ROM vuint8 resourceMask, BYTEPTR seed )` (6.5.3)

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

`vuint8 **ApplXcpUnlock**( MEMORY_ROM vuint8 *key, MEMORY_ROM vuint8 length )` (6.5.4)

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

<u>Annotation for the usage of CANape</u>

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 is locked.

<u>Example Implementation for SEEDKEY1.DLL</u>

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

Example

FILE SEEDKEY1.H

```
#ifndef _SEEDKEY_H_

#define _SEEDKEY_H_
#ifndef DllImport
#define DllImport   __declspec(dllimport)
#endif
#ifndef 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 7.2.2.1 'Table of Checksum Calculation Methods'.

If checksum calculation is enabled the background task

`vuint8 **XcpBackground**( void )` (6.2.4)

has to be called cyclically.

3.7 Memory Protection (XCP Professional only)

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

`vuint8 **ApplXcpCheckWriteAccess**( MTABYTEPTR addr, vuint8 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

`vuint8 **ApplXcpCheckReadAccess**( MTABYTEPTR addr, vuint8 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:

`vuint8 **ApplXcpCheckDAQAccess**( DAQBYTEPTR addr, vuint8 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.

3.8 Event Codes

The slave device may report events to the master device by sending asynchronous event packets (EV), which contain event codes, to the master device. The transmission is not guaranteed due to 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 **XcpSendEvent**( vuint8 evc, MEMORY_ROM BYTEPTR c, vuint8 len )` (6.2.5)

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.

3.9 Service Request Messages (XCP Professional only)

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` **`ApplXcpUserService`** `( MEMORY_ROM vuint8 c )` (6.2.6)

`void` **`ApplXcpPrint`** `( MEMORY_ROM vuint8 *str )` (6.2.7)

Refer to 7.2.1 for the configuration of the service request message.

3.10 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:

`vuint8` **`ApplXcpUserService`** `( MEMORY_ROM BYTEPTR pCmd )` (6.5.11)

3.11 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:

`vuint8` **`ApplXcpTLService`** `( MEMORY_ROM BYTEPTR pCmd )` (6.4.6)

3.12 Synchronous Data Transfer

3.12.1 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 `XcpEvent(x)` has to be called cyclically for each event channel with the corresponding event channel number as parameter. The application has to ensure that `XcpEvent` is called with the correct cycle time, which is defined in the MCS. Note that the event channel numbers have to start at 0 and have to be continuous.

The ECU automatically transmits the current value of the measurement objects via messages to the MCS, when the function `XcpEvent` 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.

<u>Annotation for the usage of CANape</u>

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

3.12.2 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 require higher precision timestamps. In such cases, ECU generated timestamps are recommended.

For the configuration of the DAQ time stamped mode refer to chapter 7.2.7 (Configuration of the DAQ Time Stamped Mode).

3.12.3 Power-Up Data Transfer (XCP Professional only)

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.

`vuint8` **`ApplXcpDaqResume`** `( tXcpDaq * daq )` (6.5.21)

`void` **`ApplXcpDaqResumeStore`** `( MEMORY_ROM tXcpDaq * daq )` (6.5.22) `void` **`ApplXcpDaqResumeClear`** `( void )` (6.5.23)

`vuint8` **`ApplXcpCalResumeStore`** `( void )` (6.5.24)

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

<u>Annotation for the usage of CANape</u>

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

3.12.4 Data Stimulation (STIM) (XCP Professional only)

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

The STIM processor buffers incoming data stimulation packets. When an event occurs (`XcpEvent` 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_DAQ` and `XCP_ENABLE_STIM` have to be defined.

3.12.5 Bypassing (XCP Professional only)

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.

3.12.6 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_DAQ_PROCESSOR_INFO`)

> general information on DAQ processing resolution
(enabled with `XCP_ENABLE_DAQ_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

3.12.7 Event Channel Plug & Play Mechanism

The XCP Protocol Layer supports a plug & play mechanism that allows the MCS to automatically detect the available event channels in the slave.

Please refer to chapter 7.2.6 (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 `XcpEvent(x)` function call in ECU source code.

3.13 The Online Data Calibration Model

3.13.1 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:

```
    vuint8 ApplXcpGetCalPage ( vuint8 segment, vuint8 mode )    (6.5.26)
    vuint8 ApplXcpSetCalPage ( vuint8 segment, vuint8 page, vuint8
                mode )                                           (6.5.27)
```

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.

3.13.2 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 7.2.9 (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_ENBALE_PAGE_INFO`.

3.13.3 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:

```
    vuint8 ApplXcpCopyCalPage( vuint8 srcSeg,  vuint8 srcPage,
                               vuint8 destSeg, vuint8 destPage )   (6.5.28)
```

3.13.4 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 ApplXcpSetFreezeMode( vuint8 segment, vuint8 mode )     (6.5.29)
vuint8 ApplXcpGetFreezeMode( vuint8 segment )                (6.5.30)
```

3.14 Flash Programming (XCP Professional only)

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.

3.14.1 Flash Programming by the ECU's Application

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

The flash programming is done with the following XCP commands PROGRAM_START, PROGRAM_RESET, PROGRAM_CLEAR, PROGRAM, PROGRAM_NEXT, PROGRAM_MAX, PROGRAM_RESET, PROGRAM_FORMAT[1], PROGRAM_VERIFY[2].

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

```
vuint8 ApplXcpProgramStart( void )                           (6.5.18)
vuint8 ApplXcpFlashClear( MTABYTEPTR a, vuint32 size )       (6.5.19)
vuint8 ApplXcpFlashProgram( MEMORY_ROM BYTEPTR data,
                            MTABYTEPTR a, vuint8 size )      (6.5.20)
```

The flash programming is enabled with the switch `XCP_ENABLE_PROGRAM`.

<u>Annotation for the usage of CANape</u>

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.

[1] Command not supported
[2] Command not supported

3.14.1.1 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 7.2.8 (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`.

3.14.2 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

```
vuint8 ApplXcpDisableNormalOperation( MTABYTEPTR a, vuint16
       size )
```
(6.5.15)

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 Graph 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 Graph.

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

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

3.14.3 Flash Programming Write Protection

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

```
vuint8 ApplXcpCheckProgramAccess( MTABYTEPTR addr, vuint8 size )
```
(6.5.10)

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

3.15 EEPROM Access (XCP Professional only)

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:

```
vuint8 ApplXcpCheckReadEEPROM
           ( MTABYTEPTR addr, vuint8 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

```
vuint8 ApplXcpCheckWriteEEPROM
           ( MTABYTEPTR addr, vuint8 size, MEMORY_ROM BYTEPTR data
           )
```
(6.5.6)

3.16 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 should be removed in order to save code space.

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

```
void XcpMemCpy( DAQBYTEPTR dest,
                MEMORY_ROM DAQBYTEPTR src, vuint16 n )          (6.6.1)
```

```
void XcpMemSet( BYTEPTR p, vuint16 n, vuint8 b )                 (6.6.2)
```

```
static void XcpMemClr( BYTEPTR p, vuint16 n )                    (6.6.3)
```

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

The transmission of data transfer objects (DTO) could also be optimized e.g. by using DMA. Therefore the following function has to be overwritten

```
void XcpSendDto( const xcpDto_t *dto )                           (6.6.4)
```

The above listed functions can be overwritten by defining a macro with the functions name that is included in the XCP Protocol Layer component.

3.18 Interrupt Locks

The functions `XcpEvent`, `XcpSendCallBack`, `XcpBackground` and `XcpCommand` are not reentrant. If one of these functions may interrupt one of the others, the functions or macros `ApplXcpInterruptEnable` (6.4.4) and `ApplXcpInterruptDisable` (6.4.5) have to be defined to protect critical sections in the code from being interrupted. The XCP Protocol Layer will not nest the sections with disabled interrupts. The time periods are as short as possible, but note that `ApplXcpSend` is called with disabled interrupts!

3.19 Accessing internal data

The function

```
void XcpGetXcpDataPointer ( RAM tXcpData ** pXcpData )           (6.2.10)
```

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.

3.20 En- / Disabling the XCP module

The function

```
void XcpControl ( vuint8 command )                               (6.2.11)
```

can be used to en- or disable the XCP module during run time. Thus the XCP functionality can be controlled by the application. The parameter “command” is either `kXcpControl_Disable` to disable the Xcp or `kXcpControl_Enable` to enable it again.

Note

Please note that when **`XcpControl`** is called all APIs are disabled but the state of the XCP remains. Thus a running DAQ list is continued after the XCP is enabled again. If you want to prevent this, don't use **`XcpControl`** to enable the XCP again but use the Xcp_Init and <BusXcp>_Init functions to initialize the XCP.

3.21 Support for address doubling in XCP for DSP micros

In order to support DSP µC that do not support Byte access the XCP provides a mechanism called "address doubling". If this feature is used all addresses received from the tool must be doubled in order to access the physical address. CANape does this automatically if a coff file for TMS320 is used as map file. If other tools are used this must be done manually in the a2l file. This also means that the addresses for user callbacks are doubled and must be divided by 2 to access the physical address. With an even virtual address the high byte of the physical address is accessed. With an odd virtual address the low byte of the physical address is accessed.

The affected APIs are:

- `ApplXcpFlashClear (6.5.19)`
- `ApplXcpFlashProgram (6.5.20)`
- `ApplXcpCheckReadAccess (6.5.8)`
- `ApplXcpCheckProgramAccess (6.5.10)`
- `ApplXcpDisableNormalOperation (6.5.15)`
- `ApplXcpCheckWriteAccess (6.5.7)`
- `ApplXcpCheckReadEEPROM (6.5.5)`
- `ApplXcpCheckWriteEEPROM (6.5.6)`
- `ApplXcpCheckDAQAccess (6.5.9)`

APIs which are not affected:

- `ApplXcpReadChecksumValue (6.5.31)`
- `ApplXcpGetSeed (6.5.3)`
- `ApplXcpUnlock (6.5.4)`
- `ApplXcpGetIdData (6.5.2)`
- `ApplXcpUserService (6.5.11)`

- `ApplXcpOpenCmdIf (6.5.12)`
- `ApplXcpWrite (6.5.33)`
- `ApplXcpRead (6.5.32)`

The following features/switches are not supported if “address doubling is used”:

- `XCP_ENABLE_CALIBRATION_MEM_ACCESS_BY_APPL`
- `XCP_ENABLE_MODIFY_BITS`

Also character arrays can not be measured as there is no way for XCP to know that it is reading a character array.

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		
`XcpProf.c`	XCP Professional source code. This file **must not** be changed by the user!	🚫
`XcpProf.h`	API of XCP Professional. This file **must not** be changed by the user!	🚫
`_xcp_appl.c`	Template that contains the application callback functions of the XCP Protocol Layer. It is just an example and has to be customized.	✎
`v_def.h`	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 GENy. If no generation tool or if CANgen is used the XPC Protocol Layer has to be customized manually. In this case the following files will be available as template.

Files generated by GENy		
`xcp_cfg.h`	XCP Protocol Layer configuration file.	🚫
`xcp_par.c`	Parameter definition for the XCP Protocol Layer.	🚫
`xcp_par.h`	External declarations for the parameters.	🚫
`v_cfg.h`	General Vector configuration file for platform specifics.	🚫
`v_inc.h`	General header for including the Vector CANbedded stacks headers.	🚫

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

4.2 Files of XCP Basic

The XCP Protocol Layer consists of the following files:

Files of the XCP Protocol Layer		
`XcpBasic.c`	XCP Basic source code. This file **must not** be changed by the application!	🚫
`XcpBasic.h`	API of XCP Basic. This file **must not** be changed by the application!	🚫
`xcp_cfg.h`	Configuration file template for the XCP Protocol Layer. It is just an example and has to be customized.	✎

`xcp_par.c`	Template with parameter definitions for the XCP Protocol Layer. It is just an example and has to be customized
`xcp_par.h`	Template with external declarations for the parameters. It is just an example and has to be customized

4.3 Version changes

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

4.4 Integration of XCP into the Application

4.4.1 Integration of XCP on CAN (XCP Professional only)

The Vector CANbedded stack includes optionally XCP on CAN, which comprises the XCP Protocol Layer in conjunction with the XCP on CAN Transport Layer and the CAN-Driver. Note that the CAN-Driver, which is distributed as a separate product, is only partly part of XCP on CAN.

The following figure shows the interface between XCP on CAN and the application:

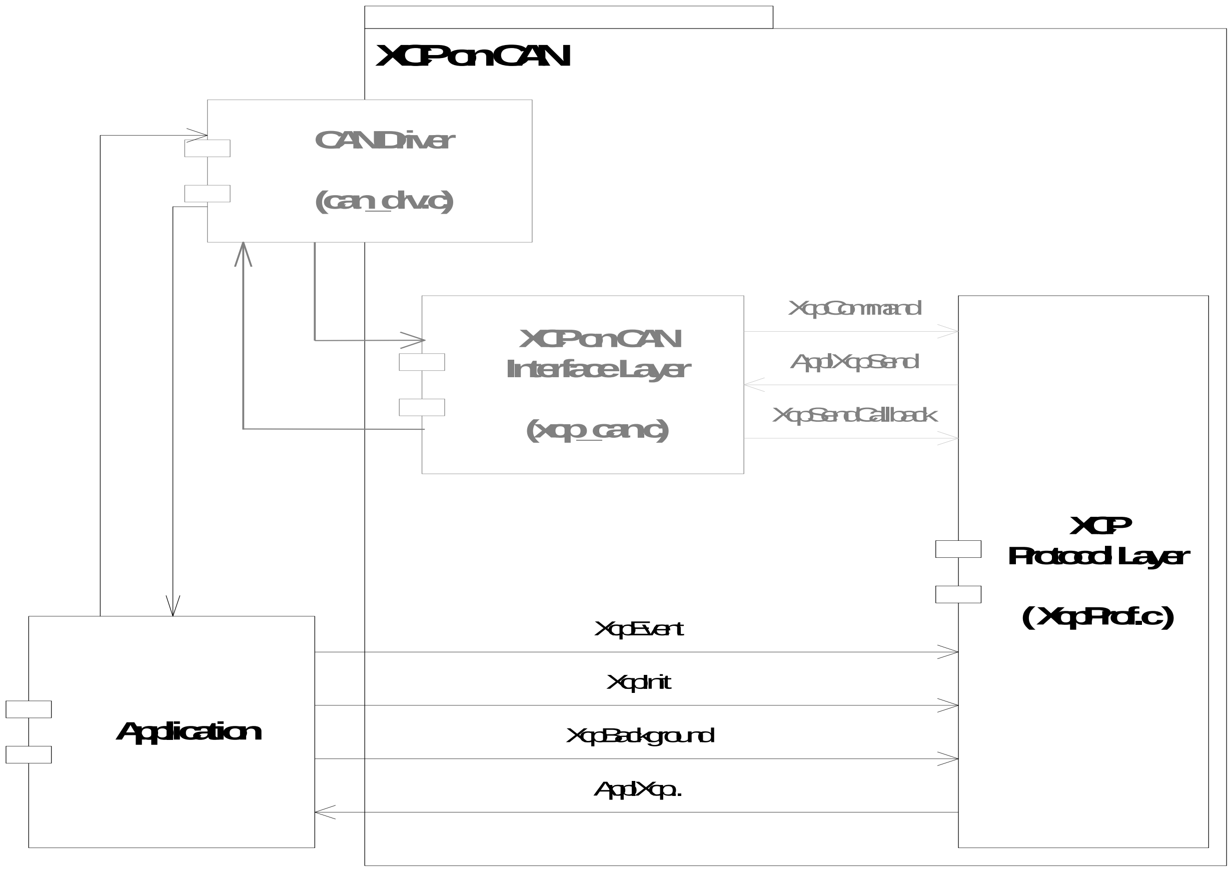


Figure 4-1 Integration of XCP on CAN into the application

	Practical Procedure The integration of XCP on CAN can be done by following these steps:

	1. Configure XCP on CAN in the generation tool GENy and generate. 2. Include the include header file `v_inc.h` into all modules that access the XCP on CAN services or provide services that XCP on CAN uses. 3. Add all source files and generated source files in the make file and link it together with the data link layer and the application. 4. Initialize the data link layer after each reset during start-up before initializing XCP on CAN (interrupts have to be disabled until the complete initialization procedure is done) by calling `XcpInit`. 5. If required call the background function `XcpBackground` cyclically. 6. Integrate the desired XCP on CAN services into your application. Call especially the function `XcpEvent(channel)` cyclic with the appropriate cycle time and channel number. The XCP on CAN sources must not be changed for the integration into the application.

4.4.2 Integration with a Proprietary XCP Transport Layer

The XCP Protocol Layer needs a XCP Transport Layer to transmit and receive XCP protocol messages on the communication link (CAN, FlexRay, Ethernet, SxI, …) that is used. The free Vector XCP Protocol Layer implementation does not include the XCP Transport Layer, which typically is strongly ECU dependant. However the Vector XCP on CAN software components already includes the XCP Transport Layer for CAN.

The following figure shows the interface between the transport layer and the protocol layer.

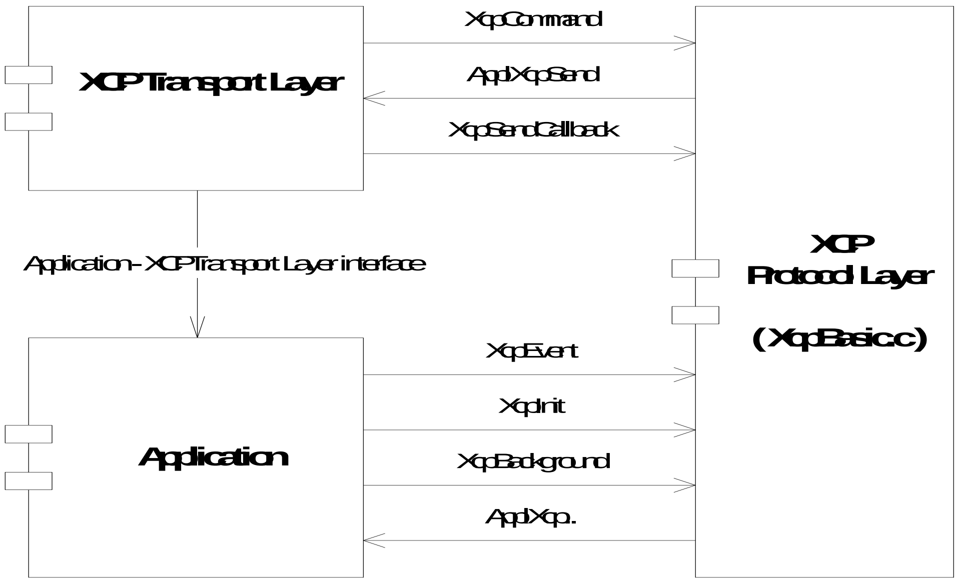


Figure 4-2 Integration of XCP with a proprietary XCP Transport Layer

The transport layer driver has to notify the protocol layer after reception of a XCP protocol message by calling the protocol layer function `XcpCommand()`.

The protocol layer will use the function `ApplXcpSend()` of the transport layer to transmit a command response message or a data acquisition message.

After the message has been transmitted successfully, the transport layer has to call the function `XcpSendCallBack()` of the protocol layer to indicate this.

The functions `XcpInit()`, `XcpEvent()` and `XcpBackground()` are called from the ECU's application program.

The function `ApplXcpGetPointer()` is used by the protocol layer to convert a 32 Bit address with an address extension to a valid pointer.

Depending on the optional features that can be enabled upon demand further application callback functions are necessary. All application functions are indicated in Figure 4-2 by their prefix `ApplXcp`….

Example

The following C pseudo code example shows the required software handshake between the protocol layer and the transport layer. The example uses a simple transport layer definition where the length of the protocol message is transmitted in the first byte of the protocol packet:

```
/* Initialization */
XcpInit();

/* Main Loop */
for (;;) {

  /* Packet received */
  if (Message received) {
    XcpCommand(&ReceiveBuffer[1]);
  }

  /* Transmit Message Buffer available */
  if (Message transmitted) {
    XcpSendCallBack();
  }

  /* Background Processing */
  XcpBackground();
}

/* Transmit Function */
void ApplXcpSend(vuint8 len, MEMORY_ROM BYTEPTR msg ) {
  TransmitBuffer[0] = len; /* This is transport layer
specific */
  memcpy(&TransmitBuffer[1],msg,len);
  Transmit(TransmitBuffer);
}


/* Pointer Conversion */
MTABYTEPTR ApplXcpGetPointer( vuint8 addr_ext, vuint32 addr ) {
  Return (BYTE*)addr;
}
```

4.4.3 Motorola HC12 with CAN Transport Layer

See the application note “AN-IMC-1-007_Integration_of_the_Vector_XCP_Driver with_a_free_CAN_Driver_v1.0.0_EN.pdf” which explains in detail how to integrate the Vector basic XCP driver into an HC12 microcontroller with an existing CAN driver.

5 Feature List

This general feature list describes the overall feature set of the XCP Protocol Layer. Not all of these features are available in XCP Basic. Please also refer to 9.2 “Limitations of XCP Basic”.

Description of the XCP functionality	Version	Functions
Initialization		
Initialization	Prof, Basic	`XcpInit` `ApplXcpInit`
Task		
Background task	Prof, Basic	`ApplXcpBackground` `XcpBackground`
XCP Command Processor		
Command Processor	Prof, Basic	`XcpCommand`
Transmission and Confirmation of XCP Packets	Prof, Basic	`ApplXcpSend` `XcpSendCallBack`
Transmission of Response packets	Prof, Basic	`XcpSendCrm`
Transmission of XCP Packets	Prof, Basic	`ApplXcpSendStall` `ApplXcpSendFlush`
XCP Commands		
Get MAP filenames	Prof, Basic	`ApplXcpGetIdData`
Seed & Key	Prof, Basic	`ApplXcpGetSeed` `ApplXcpUnlock`
Short Download	Prof	-
Modify Bits	Prof	-
Write DAQ Multiple	Prof	`ApplXcpCheckDAQAccess`
Transport Layer Command	Prof	-
Open Command Interface	Prof	-
User command	Prof, Basic	`ApplXcpUserService`
Data Acquisition (DAQ)		
Synchronous Data Acquisition and Stimulation	Prof, Basic	`XcpEvent` `ApplXcpCheckDAQAccess`
DAQ Timestamp	Prof, Basic	`ApplXcpGetTimestamp`

Resume Mode	Prof	`ApplXcpDaqResume`
		`ApplXcpDaqResumeStore`
		`ApplXcpDaqResumeClear`
		`ApplXcpCalResumeStore`

Online Data Calibration		
Calibration page switching	Prof, Basic	`ApplXcpGetCalPage` `ApplXcpSetCalPage`
Copy calibration page	Prof, Basic	`ApplXcpCopyCalPage`
Freeze Mode	Prof, Basic	`ApplXcpSetFreezeMode` `ApplXcpGetFreezeMode`

Boot loader Download		
Disable normal operation of ECU	Prof	`ApplXcpDisableNormalOperation`
Start of the boot loader	Prof	`ApplXcpStartBootLoader`
Flash Programming		
Reset of ECU	Prof	`ApplXcpReset`
Clear flash memory	Prof	`ApplXcpFlashClear`
Prepare flash programming	Prof	`ApplXcpProgramStart`
Program flash memory	Prof	`ApplXcpFlashProgram`
Special Features		
Interrupt Control	Prof, Basic	`ApplXcpInterruptEnable` `ApplXcpInterruptDisable`
Event Codes	Prof	`XcpSendEvent`
Service Request Packets	Prof	`XcpPutchar` `XcpPrint`
Disconnect XCP	Prof, Basic	`ApplXcpDisconnect`
Pointer conversion	Prof, Basic	`ApplXcpGetPointer`
EEPROM access	Prof	`ApplXcpCheckReadEEPROM` `ApplXcpCheckWriteEEPROM`
Write protection	Prof	`ApplXcpCheckWriteAccess`
Read protection	Prof	`ApplXcpCheckReadAccess`
Overwriteable macros	Prof, Basic	`XcpMemCpy` `XcpMemSet` `XcpMemClr` `XcpSendDto`
En- / Disabling XCP module	Prof	`XcpControl`
Access to internal data	Prof	`XcpGetXcpDataPointer`
En-/Disable Calibration	Prof	-
Programming Write Protection	Prof	`ApplXcpCheckProgramAccess`

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:

```
V_MEMROM0 MEMORY_ROM vuint8 kCp_XcpMainVersion =
      (vuint8)(CP_XCP_VERSION >> 8);
V_MEMROM0 MEMORY_ROM vuint8 kCp_XcpSubVersion  =
      (vuint8)(CP_XCP_ VERSION);
V_MEMROM0 MEMORY_ROM vuint8 kCp_XcpReleaseVersion =
      (vuint8)(CP_XCP_RELEASE_VERSION);
```

Example

Version 1.00.00 is registered as:

```
kCp_XcpMainVersion    = 0x01;
kCp_XcpSubVersion     = 0x00;
kCp_XcpReleaseVersion = 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` **`XcpGetVersionInfo`** `(Std_VersionInfoType *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 XcpInit: Initialization of the XCP Protocol Layer

XcpInit

Prototype	
Single Channel	
Single Receive Channel	void **XcpInit** (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.	

6.2.2 XcpEvent: Handling of a data acquisition event channel

XcpEvent

Prototype	
Single Channel	
Single Receive Channel	vuint8 XcpEvent (vuint8 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	
vuint8	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 XcpEvent 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 XcpEvent calls have to be placed at the appropriate locations in the ECU's program: xcpEvent (0); /* 10ms cycle */ xcpEvent (1); /* 100ms cycle */ xcpEvent (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)

6.2.3 XcpStimEventStatus: Check data stimulation events

XcpStimEventStatus

Prototype	
Single Channel	
Single Receive Channel	vuint8 **XcpStimEventStatus** (vuint8 event, vuint8 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	
`vuint8`	`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)	

6.2.4 XcpBackground: Background calculation of checksum

XcpBackground

Prototype	
Single Channel	
Single Receive Channel	vuint8 **XcpBackground** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
`vuint8`	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 `XcpBackground`. Therefore `XcpBackground` should be called periodically whenever the ECU's CPU is idle.	
Particularities and Limitations	
> The XCP Protocol Layer has been initialized correctly > Call context: Task level	

6.2.5 XcpSendEvent: Transmission of event codes

XcpSendEvent

Prototype	
Single Channel	
Single Receive Channel	void **XcpSendEvent** (vuint8 evc, MEMORY_ROM BYTEPTR c, vuint8 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 3.8 Event Codes.
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_SEND_EVENT has to be defined > Call context: Task and interrupt level

6.2.6 XcpPutchar: Put a char into a service request packet

XcpPutchar

Prototype	
Single Channel	
Single Receive Channel	void **XcpPutchar** (MEMORY_ROM vuint8 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	
> The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > The switch `XCP_ENABLE_SERV_TEXT_PUTCHAR` has to be defined > Call context: Task and interrupt level (not reentrant)	

6.2.7 XcpPrint: Transmission of a service request packet

XcpPrint

Prototype	
Single Channel	
Single Receive Channel	void **XcpPrint** (MEMORY_ROM vuint8 *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 `str` is sent via service request packets. The string has to be terminated by 0x00.	
Particularities and Limitations	
> The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > The switch `XCP_ENABLE_SERV_TEXT_PRINT` has to be defined > Call context: Task and interrupt level (not reentrant)	

6.2.8 XcpDisconnect: Disconnect from XCP master

XcpDisconnect

Prototype	
Single Channel	
Single Receive Channel	void **XcpDisconnect** (void)
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	
> The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > Call context: Task and interrupt level (not reentrant)	

6.2.9 XcpSendCrm: Transmit response or error packet

XcpSendCrm

Prototype	
Single Channel	
Single Receive Channel	void **XcpSendCrm** (void)
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	
> 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)	

6.2.10 XcpGetXcpDataPointer: Request internal data pointer

XcpGetXcpDataPointer

Prototype	
Single Channel	
Single Receive Channel	void **XcpGetXcpDataPointer** (RAM 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 `XCP_ENABLE_GET_XCP_DATA_POINTER` has to be defined	

6.2.11 XcpControl: En- / Disable the XCP module

XcpControl

Prototype	
Single Channel	
Single Receive Channel	void **XcpControl** (vuint8 command)
Multi Channel	
Indexed	not supported

Code replicated	not supported
Parameter	
`command`	parameter to either en- or disable the module kXcpControl_Disable: disable the module kXcpControl_Enable: enable the module
Return code	
-	-
Functional Description	
With this function the whole module can be en- or disabled. After initialization the module is enabled. A call with parameter kXcpControl_Enable does not lead to any changed behavior. After call with parameter kXcpControl_Disable each function either called by the application or by the transport layer is directly left without any handling. Thus this function can be used to disable the XCP functionality during runtime.	
Particularities and Limitations	
> The switch `XCP_ENABLE_CONTROL` has to be defined	

6.2.12 XcpGetVersionInfo: Request module version information

XcpGetVersionInfo

Prototype	
Single Channel	
Single Receive Channel	void **XcpGetVersionInfo** (Std_VersionInfoType *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	
With this service it is possible to get the version information of this software module.	
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 XcpCommand: Evaluation of XCP packets and command interpreter

XcpCommand

Prototype	
Single Channel	
Single Receive Channel	void **XcpCommand** (MEMORY_ROM vuint32* 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	
> The XCP Protocol Layer has to be initialized correctly. > Call context: Task and interrupt level (not reentrant)	

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

XcpSendCallBack

Prototype	
Single Channel	
Single Receive Channel	vuint8 **XcpSendCallBack** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported

Parameter	
-	-
Return code	
`vuint8`	`0`: if the XCP Protocol Layer is idle (no transmit messages are pending)
Functional Description	
The XCP Protocol Layer does not call `ApplXcpSend` again, until `XcpSendCallBack` has confirmed the successful transmission of the previous message. `XcpSendCallBack` transmits pending data acquisition messages by calling `ApplXcpSend` again. Note that if `XcpSendCallBack` is called from inside `ApplXcpSend` a recursion occurs, which assumes enough space on the call stack.	
Particularities and Limitations	
> The XCP Protocol Layer has been initialized correctly. > Call context: Task and interrupt level (not reentrant)	

6.3.3 XcpGetState: Get connection state of XCP

XcpGetState

Prototype	
Single Channel	
Single Receive Channel	vuint8 **XcpGetState** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
`XCP_CONNECTED`	XCP is connected
`XCP_DISCONNECTED`	XCP is disconnected
Functional Description	
Get the connection state of the XCP Protocol Layer. 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.	
Particularities and Limitations	
> The XCP Protocol Layer has to be initialized correctly. > Call context: Task and interrupt level (not reentrant) > Enabled/Disabled by `XCP_xxx_GET_CONNECTION_STATE`	

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 ApplXcpSend: Request for the transmission of a DTO or CTO message

ApplXcpSend

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpSend** (vuint8 len, MEMORY_ROM BYTEPTR msg)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`len` `msg`	Length of message data Pointer to message
Return code	
`vuint8`	`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). `XcpSendCallBack` must be called after the successful transmission of any XCP message. The XCP Protocol Layer will not request further transmissions, until `XcpSendCallBack` has been called.	
Particularities and Limitations	
> Call context: Task and interrupt level (not reentrant) > `ApplXcpSend` is not defined as macro	

6.4.2 ApplXcpInit: Perform XCP Transport Layer initialization

ApplXcpInit

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpInit** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-

Functional Description
Initializations of the XCP Transport Layer. This function is required by XCP on CAN if no CAN transmit queue is used.
Particularities and Limitations
> Call context: Task and interrupt level (context of XcpInit) > `ApplXcpInit` is not defined as macro

6.4.3 ApplXcpBackground: XCP Transport Layer background operations

ApplXcpBackground

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpBackground** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Performs background operations of the XCP Transport Layer. This function is required by XCP on CAN if no CAN transmit queue is used.	
Particularities and Limitations	
> Call context: Task and interrupt level (context of XcpBackground) > `ApplXcpBackground` is not defined as macro	

6.4.4 ApplXcpInterruptEnable: Enable interrupts

ApplXcpInterruptEnable

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpInterruptEnable** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-

Functional Description
Enabling of the global interrupts.
Particularities and Limitations
> XCP is initialized correctly > Call context: Task and interrupt level > This function is reentrant! > The function `ApplXcpInterruptEnable` can be overwritten by the macro `ApplXcpInterruptEnable`.

6.4.5 ApplXcpInterruptDisable: Disable interrupts

ApplXcpInterruptDisable

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpInterruptDisable** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Disabling of the global interrupts.	
Particularities and Limitations	
> XCP is initialized correctly > Call context: Task and interrupt level > This function is reentrant! > The function `ApplXcpInterruptDisable` can be overwritten by the macro `ApplXcpInterruptDisable`.	

6.4.6 ApplXcpTLService: Transport Layer specific commands

ApplXcpTLService

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpTLService** (MEMORY_ROM BYTEPTR 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	
vuint8	XCP_CMD_OK : Done XCP_CMD_PENDING : Call XcpSendCrm() when done XCP_CMD_SYNTAX : Error XCP_CMD_BUSY : not executed XCP_CMD_UNKNOWN : not implemented optional command XCP_CMD_OUT_OF_RANGE : command parameters out of range
Functional Description	
Transport Layer specific 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 ApplXcpGetPointer: Pointer conversion

ApplXcpGetPointer

Prototype	
Single Channel	
Single Receive Channel	MTABYTEPTR **ApplXcpGetPointer** (vuint8 addr_ext, vuint32 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 `ApplXcpGetPointer`:
`MTABYTEPTR ApplXcpGetPointer( vuint8 addr_ext, vuint32 addr )`
`{`
`  return (MTABYTEPTR)addr;`
`}`

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

6.5.2 ApplXcpGetIdData: Get MAP filenames

ApplXcpGetIdData

Prototype	
Single Channel	
Single Receive Channel	vuint32 **ApplXcpGetIdData** (MTABYTEPTR *pData)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`pData`	Returns a pointer to a pointer of MAP file names
Return code	
`vuint32`	length of the MAP file names
Functional Description	
Returns a pointer to a pointer of MAP file names. Refer to chapter 3.4.2 (Transferring of XCP MAP Filenames).	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_VECTOR_MAPNAMES` has to be defined	

6.5.3 ApplXcpGetSeed: Generate a seed

ApplXcpGetSeed

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpGetSeed** (MEMORY_ROM vuint8 resource, BYTEPTR seed)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`Resource`	Resource for which the seed has to be generated XCP Professional and XPC Basic `RM_CAL_PAG` : to unlock the resource calibration/paging `RM_DAQ` : to unlock the resource data acquisition XCP Professional only `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	
`vuint8`	The length of the generated seed that is returned by *seed*.
Functional Description	
Generate a seed for the appropriate resource. The seed has a maximum length of MAX_CTO-2 bytes.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_SEED_KEY` has to be defined	

6.5.4 ApplXcpUnlock: Valid key and unlock resource

ApplXcpUnlock

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpUnlock** (MEMORY_ROM vuint8 *key, MEMORY_ROM vuint8 length)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`key`	Pointer to the key.
`length`	Length of the key.

Return code	
vuint8	<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 ApplXcpCheckReadEEPROM: Check read access from EEPROM

ApplXcpCheckReadEEPROM

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCheckReadEEPROM** (MTABYTEPTR addr, vuint8 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	
`vuint8`	`0` : This is not EEPROM `1` : Read from EEPROM
Functional Description	
Checks whether the address lies within the EEPROM memory or in the RAM area. If the area is within the EEPROM area `size` data byte are read from `addr` and written to `data`.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_READ_EEPROM` has to be defined	

6.5.6 ApplXcpCheckWriteEEPROM: Check write access to the EEPROM

ApplXcpCheckWriteEEPROM

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCheckWriteEEPROM** (MTABYTEPTR addr, vuint8 size, MEMORY_ROM 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 `addr` is on the EEPROM this data is written to `addr`)
Return code	
`vuint8`	`XCP_CMD_OK` : EEPROM written `XCP_CMD_DENIED` : This is not EEPROM `XCP_CMD_PENDING` : EEPROM write in progress, call `XcpSendCrm` 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 `XcpSendCrm` function must be called. `XcpSendCrm` 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 ApplXcpCheckWriteAccess: Check address for valid write access

ApplXcpCheckWriteAccess

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCheckWriteAccess** (MTABYTEPTR address, vuint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported

Parameter	
`address`	address
`size`	number of bytes
Return code	
`vuint8`	`0 :` if access is denied `>= 1 :` 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	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_WRITE_PROTECTION` has to be defined > Can be overwritten by the macro `ApplXcpCheckWriteAccess`	

6.5.8 ApplXcpCheckReadAccess: Check address for valid read access

ApplXcpCheckReadAccess

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCheckReadAccess** (MTABYTEPTR address, vuint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`address`	address
`size`	number of bytes
Return code	
`vuint8`	`0 :` if access is denied `>= 1 :` 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 `ApplXcpCheckReadAccess`	

6.5.9 ApplXcpCheckDAQAccess: Check address for valid read or write access

ApplXcpCheckDAQAccess

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCheckDAQAccess** (DAQBYTEPTR address, vuint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`address`	address
`size`	number of bytes
Return code	
`vuint8`	`XCP_CMD_DENIED` : if access is denied `XCP_CMD_OK` : 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 ApplXcpCheckProgramAccess: Check address for valid write access

ApplXcpCheckProgramAccess

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCheckProgramAccess** (MTABYTEPTR address, vuint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`address`	address
`size`	number of bytes
Return code	
`vuint8`	`0` : if access is denied `>= 1` : 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
> 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 `ApplXcpCheckWriteAccess`

6.5.11 ApplXcpUserService: User defined command

ApplXcpUserService

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpUserService** (MEMORY_ROM BYTEPTR pCmd)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`pCmd`	Pointer to XCP command packet
Return code	
`vuint8`	`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 3.10 User Defined Command.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_USER_COMMAND` has to be defined	

6.5.12 ApplXcpOpenCmdIf: XCP command extension interface

ApplXcpOpenCmdIf

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpOpenCmdIf** (MEMORY_ROM BYTEPTR pCmd BYTEPTR pRes, BYTEPTR pLength)
Multi Channel	
Indexed	not supported

Code replicated	not supported
Parameter	
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	
vuint8	XCP_CMD_OK : Done XCP_CMD_PENDING : Call XcpSendCrm() 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	
> XCP is initialized correctly > Call context: Task and interrupt level > The switch XCP_ENABLE_OPENCMDIF has to be defined	

6.5.13 ApplXcpSendStall: Resolve a transmit stall condition

ApplXcpSendStall

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpSendStall** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
vuint8	0 : if not successful > 0 : successful
Functional Description	
Resolve a transmit stall condition in XcpPutchar or XcpSendEvent.	
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 ApplXcpSendStall()	

6.5.14 ApplXcpSendFlush: Flush transmit buffer

ApplXcpSendFlush

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpSendFlush** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
Flush the transmit buffer if there is one implemented in `ApplXcpSend`. This function can be overwritten by a macro.	
Particularities and Limitations	
> The function can be overwritten by the macro `ApplXcpSendFlush()`	

6.5.15 ApplXcpDisableNormalOperation: Disable normal operation of the ECU

ApplXcpDisableNormalOperation

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpDisableNormalOperation** (MTABYTEPTR a, vuint16 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	
`vuint8`	`XCP_CMD_OK` : download of flash kernel confirmed `XCP_CMD_DENIED` : download of flash kernel refused
Functional Description	
Prior to the flash kernel download has the ECU's normal operation to be stopped in order to avoid misbehavior due to data inconsistencies.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_BOOTLOADER_DOWNLAOD` has to be defined	

6.5.16 ApplXcpStartBootLoader: Start of boot loader

ApplXcpStartBootLoader

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpStartBootLoader** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
`vuint8`	This function should not return. `0` : negative response `> 0` : positive response
Functional Description	
Start of the boot loader.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_BOOTLOADER_DOWNLAOD` has to be defined	

6.5.17 ApplXcpReset: Perform ECU reset

ApplXcpReset

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpReset** (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
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_PROGRAM` has to be defined

6.5.18 ApplXcpProgramStart: Prepare flash programming

ApplXcpProgramStart

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpProgramStart** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
–	-
Return code	
`vuint8`	`XCP_CMD_OK` : Preparation done `XCP_CMD_PENDING` : Call XcpSendCrm() when done `XCP_CMD_ERROR` : Flash programming not possible
Functional Description	
Prepare the ECU for flash programming.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switch `XCP_ENABLE_PROGRAM` has to be defined	

6.5.19 ApplXcpFlashClear: Clear flash memory

ApplXcpFlashClear

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpFlashClear** (MTABYTEPTR address, vuint32 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`address`	Address
`size`	Size

Return code	
vuint8	XCP_CMD_OK : Flash memory erase done XCP_CMD_PENDING : Call XcpSendCrm() 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	
> XCP is initialized correctly and in connected state > The switch XCP_ENABLE_PROGRAM has to be defined	

6.5.20 ApplXcpFlashProgram: Program flash memory

ApplXcpFlashProgram

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpFlashProgram** (MEMORY_ROM BYTEPTR data, MTABYTEPTR address, vuint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
data	Pointer to data
address	Address
size	Size
Return code	
vuint8	XCP_CMD_OK : Flash memory programming finished XCP_CMD_PENDING :Flash memory programming in progress. XcpSendCrm 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	

6.5.21 ApplXcpDaqResume: Resume automatic data transfer

ApplXcpDaqResume

Prototype		
Single Channel		
Single Receive Channel	vuint8 **ApplXcpDaqResume** (tXcpDaq * daq)	
Multi Channel		
Indexed	not supported	
Code replicated	not supported	
Parameter		
`daq`	Pointer to dynamic DAQ list structure	
Return code		
`vuint8`	0 : >0 :	failed 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 `ApplXcpDaqResumeStore(..)` 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		

6.5.22 ApplXcpDaqResumeStore: Store DAQ lists for resume mode

ApplXcpDaqResumeStore

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpDaqResumeStore** (MEMORY_ROM tXcpDaq * daq)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`daq`	Pointer to dynamic DAQ list structure.
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 `ApplXcpDaqResume(..)`.	

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

6.5.23 ApplXcpDaqResumeClear: Clear stored DAQ lists

ApplXcpDaqResumeClear

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpDaqResumeClear** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
The whole dynamic DAQ list structure that had been stored in non-volatile memory within the service `ApplXcpDaqResumeStore(..)` has to be cleared.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_DAQ` and `XCP_ENABLE_DAQ_RESUME` are defined	

6.5.24 ApplXcpCalResumeStore: Store Calibration data for resume mode

ApplXcpCalResumeStore

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCalResumeStore** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
`vuint8`	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
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_DAQ` and `XCP_ENABLE_DAQ_RESUME` are defined

6.5.25 ApplXcpGetTimestamp: Returns the current timestamp

ApplXcpGetTimestamp

Prototype	
Single Channel	
Single Receive Channel	XcpDaqTimestampType **ApplXcpGetTimestamp** (void)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
`XcpDaqTimestampType`	timestamp
Functional Description	
Returns the current timestamp.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_DAQ` and `XCP_ENABLE_DAQ_TIMESTAMP` are defined > The parameter `kXcpDaqTimestampSize` defines the timestamp size. It can either be `DAQ_TIMESTAMP_BYTE, DAQ_TIMESTAMP_WORD, DAQ_TIMESTAMP_DWORD`	

6.5.26 ApplXcpGetCalPage: Get calibration page

ApplXcpGetCalPage

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpGetCalPage** (vuint8 segment, vuint8 mode)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`segment`	Logical data segment number

mode	Access mode The access mode can be one of the following values: CAL_ECU : ECU access CAL_XCP : XCP access
Return code	
vuint8	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	

6.5.27 ApplXcpSetCalPage: Set calibration page

ApplXcpSetCalPage

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpSetCalPage** (vuint8 segment, vuint8 page, vuint8 mode)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
segment	Logical data segment number
Page	Logical data page number
mode	Access mode CAL_ECU : the given page will be used by the slave device application CAL_XCP : the slave device XCP driver will access the given page Both flags may be set simultaneously or separately.
Return code	
vuint8	0 : Ok CRC_OUT_OF_RANGE : segment out of range (only one segment supported) CRC_PAGE_NOT_VALID : Selected page not available CRC_PAGE_MODE_NOT_VALID : 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

6.5.28 ApplXcpCopyCalPage: Copying of calibration data pages

ApplXcpCopyCalPage

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpCopyCalPage** (vuint8 srcSeg, vuint8 srcPage vuint8 destSeg, vuint8 destPage)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`srcSeg`	Source segment
`srcPage`	Source page
`destSeg`	Destination segment
`destPage`	Destination page
Return code	
`vuint8`	0 : Ok `XCP_CMD_PENDING` : Call XcpSendCrm() when done `CRC_PAGE_NOT_VALID` : Page not available `CRC_SEGMENT_NOT_VALID` : Segment not available `CRC_WRITE_PROTECTED` : Destination page is write protected.
Functional Description	
Copying of calibration data pages. The pages are copied from source to destination.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_PAGE_COPY` and `XCP_ENABLE_DAQ_TIMEOUT` are defined	

6.5.29 ApplXcpSetFreezeMode: Setting the freeze mode of a segment

ApplXcpSetFreezeMode

Prototype	
Single Channel	
Single Receive Channel	void **ApplXcpSetFreezeMode** (vuint8 segment, vuint8 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	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_PAGE_FREEZE` is defined	

6.5.30 ApplXcpGetFreezeMode: Reading the freeze mode of a segment

ApplXcpGetFreezeMode

Prototype	
Single Channel	
Single Receive Channel	vuint8 **ApplXcpGetFreezeMode** (vuint8 segment)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`segment`	Segment to read freeze mode
Return code	
`vuint8`	Return the current freeze mode, set by ApplXcpSetFreezeMode().
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	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_PAGE_FREEZE` is defined	

6.5.31 ApplXcpReadChecksumValue: Read a single byte from memory for checksum creation

ApplXcpReadChecksumValue

Prototype	
Single Channel	
Single Channel	tXcpChecksumAddType **ApplXcpRead** (vuint32 addr)
Multi Channel	
Indexed	not supported

Code replicated	not supported
Parameter	
`addr`	32 Bit address
Return code	
`tXcpChecksumAddType`	Value used to create checksum.
Functional Description	
Read from the memory to create checksum	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_MEM_ACCESS_BY_APPL and XCP_ENABLE_CHECKSUM` is defined	

6.5.32 ApplXcpRead: Read a single byte from memory

ApplXcpRead

Prototype	
Single Channel	
Single Channel	vuint8 **ApplXcpRead** (vuint32 addr)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`addr`	32 Bit address
Return code	
`vuint8`	Pointer to the address specified by the parameters
Functional Description	
Read a single byte from the memory.	
Particularities and Limitations	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_MEM_ACCESS_BY_APPL` is defined	

6.5.33 ApplXcpWrite: Write a single byte to RAM

ApplXcpWrite

Prototype	
Single Channel	
Single Channel	void **ApplXcpWrite** (vuint32 addr, vuint8 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	
> XCP is initialized correctly and in connected state > The switches `XCP_ENABLE_MEM_ACCESS_BY_APPL` 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 XcpMemCpy: Copying of a memory range

XcpMemCpy

Prototype	
Single Channel	
Single Receive Channel	void **XcpMemCpy** (DAQBYTEPTR dest, MEMORY_ROM DAQBYTEPTR src, vuint8 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 `XcpEvent` 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 `XcpMemCpy` is defined.

6.6.2 XcpMemSet: Initialization of a memory range

XcpMemSet

Prototype	
Single Channel	
Single Receive Channel	void **XcpMemSet** (BYTEPTR p, vuint16 n, vuint8 b)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
`p`	pointer to start address
`n`	number of data bytes
`b`	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 `XcpMemSet` is defined.	

6.6.3 XcpMemClr: Clear a memory range

XcpMemClr

Prototype	
Single Channel	
Single Receive Channel	static void **XcpMemClr** (BYTEPTR p, vuint16 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	
> The XCP Protocol Layer has been initialized correctly. > This function can be overwritten if XcpMemClr is defined.	

6.6.4 XcpSendDto: Transmission of a data transfer object

XcpSendDto

Prototype	
Single Channel	
Single Receive Channel	void **XcpSendDto** (MEMORY_ROM xcpDto_t *dto)
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
dto	pointer to data transfer object
Return code	
-	-
Functional Description	
Transmit a data transfer object (DTO).	
Particularities and Limitations	
> The XCP Protocol Layer has been initialized correctly and XCP is in connected state. > The switch XCP_ENABLE_DAQ is defined > This function can be overwritten by defining XcpSendDto.	

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].

7 Configuration of the XCP Protocol Layer

This chapter describes the common options for configuring (customizing) the XCP Protocol Layer. Please note that the XCP Professional can conveniently be configured with GENy (chapter 7.1). In this case no manual configuration has to be applied to the configuration files.
The configuration of the XCP Protocol Layer without GENy can be found in chapter 7.2.It is mainly applicable for the configuration of XCP Basic.

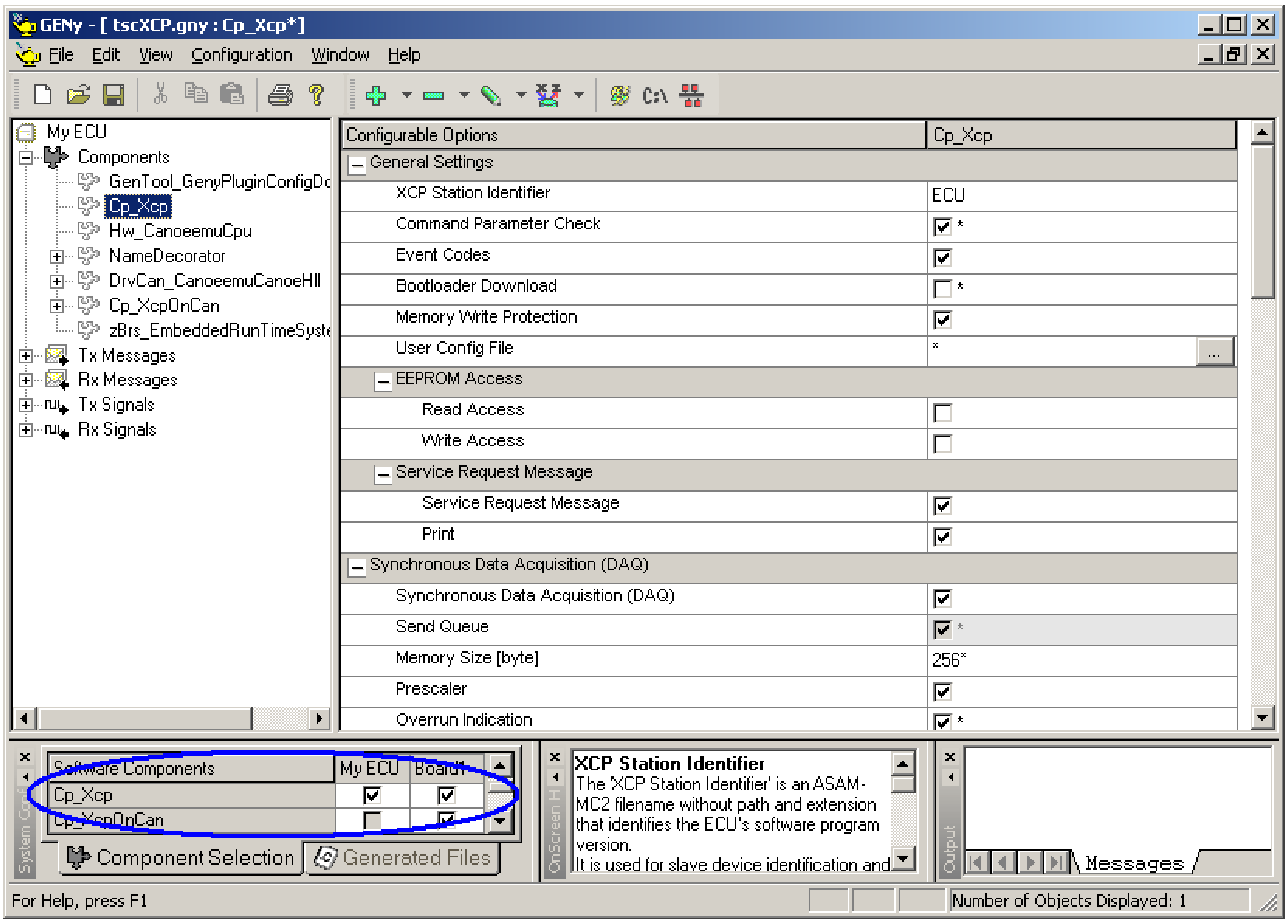
7.1 Configuration with GENy (XCP Professional only)

The XCP Protocol Layer is a higher software layer that can be configured independent of the communication system channels. Therefore in GENy the Protocol Layer component is attached to the ECU. I.e. it can be configured without associating any XCP Transport Layer in GENy.

Therefore there are no database attributes defined for the XCP Protocol Layer.

7.1.1 Component Selection

The following figure shows the selection of the XCP Protocol Layer component:



Configurable Options	Cp_Xcp
General Settings	
XCP Station Identifier	ECU
Command Parameter Check	✓ *
Event Codes	✓
Bootloader Download	☐ *
Memory Write Protection	✓
User Config File	*
EEPROM Access	
Read Access	☐
Write Access	☐
Service Request Message	
Service Request Message	✓
Print	✓
Synchronous Data Acquisition (DAQ)	
Synchronous Data Acquisition (DAQ)	✓
Send Queue	✓ *
Memory Size [byte]	256*
Prescaler	✓
Overrun Indication	✓ *

Software Components	My ECU	Board
Cp_Xcp	✓	✓
Cp_XcpOnCan	☐	✓

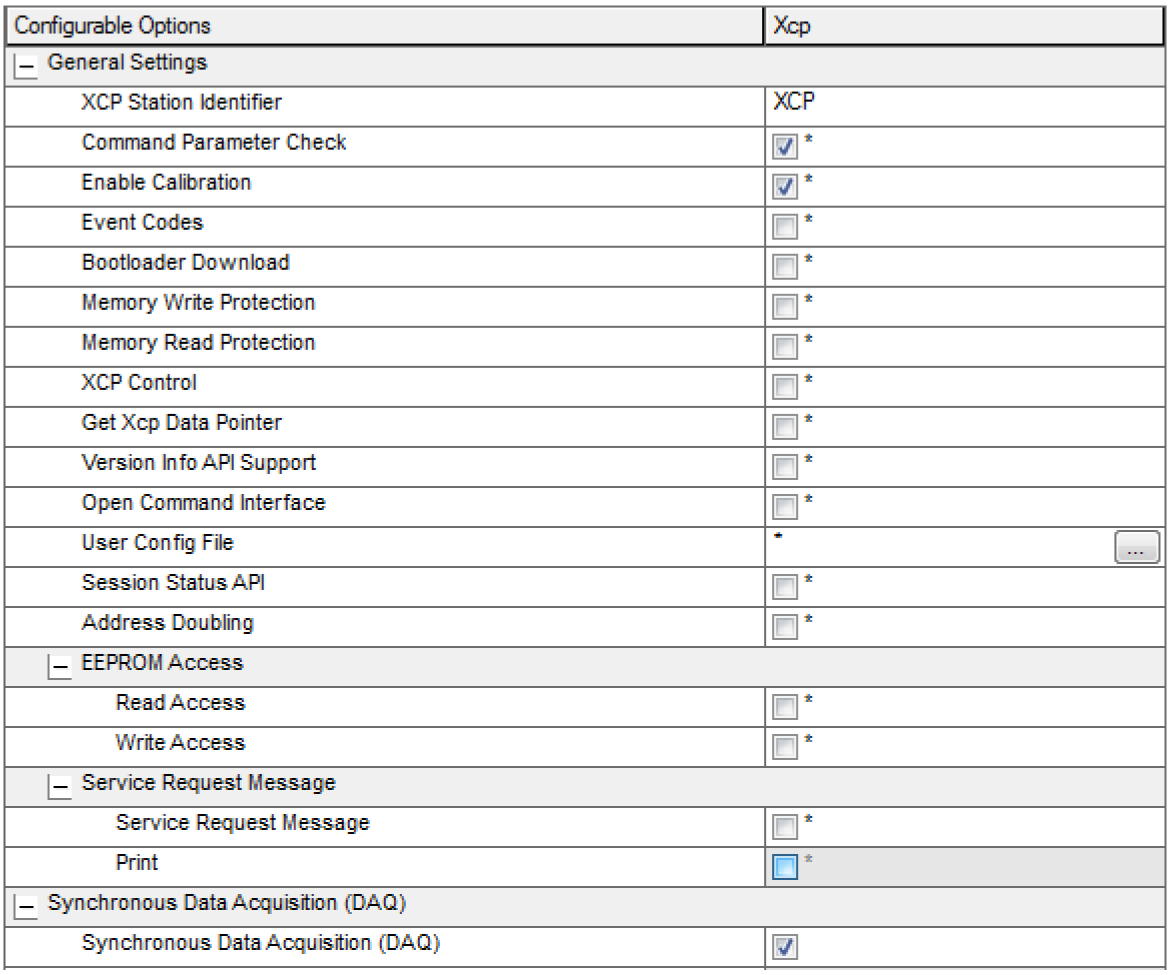
XCP Station Identifier
The 'XCP Station Identifier' is an ASAM-MC2 filename without path and extension that identifies the ECU's software program version.
It is used for slave device identification and

For Help, press F1
Number of Objects Displayed: 1

Figure 7-1 Component selection in GENy

7.1.2 Component Configuration

7.1.2.1 General Settings



Configurable Options	Xcp
General Settings	
XCP Station Identifier	XCP
Command Parameter Check	✓ *
Enable Calibration	✓ *
Event Codes	☐ *
Bootloader Download	☐ *
Memory Write Protection	☐ *
Memory Read Protection	☐ *
XCP Control	☐ *
Get Xcp Data Pointer	☐ *
Version Info API Support	☐ *
Open Command Interface	☐ *
User Config File	* ...
Session Status API	☐ *
Address Doubling	☐ *
EEPROM Access	
Read Access	☐ *
Write Access	☐ *
Service Request Message	
Service Request Message	☐ *
Print	☐ *
Synchronous Data Acquisition (DAQ)	
Synchronous Data Acquisition (DAQ)	✓

Figure 7-2 Component configuration – General settings

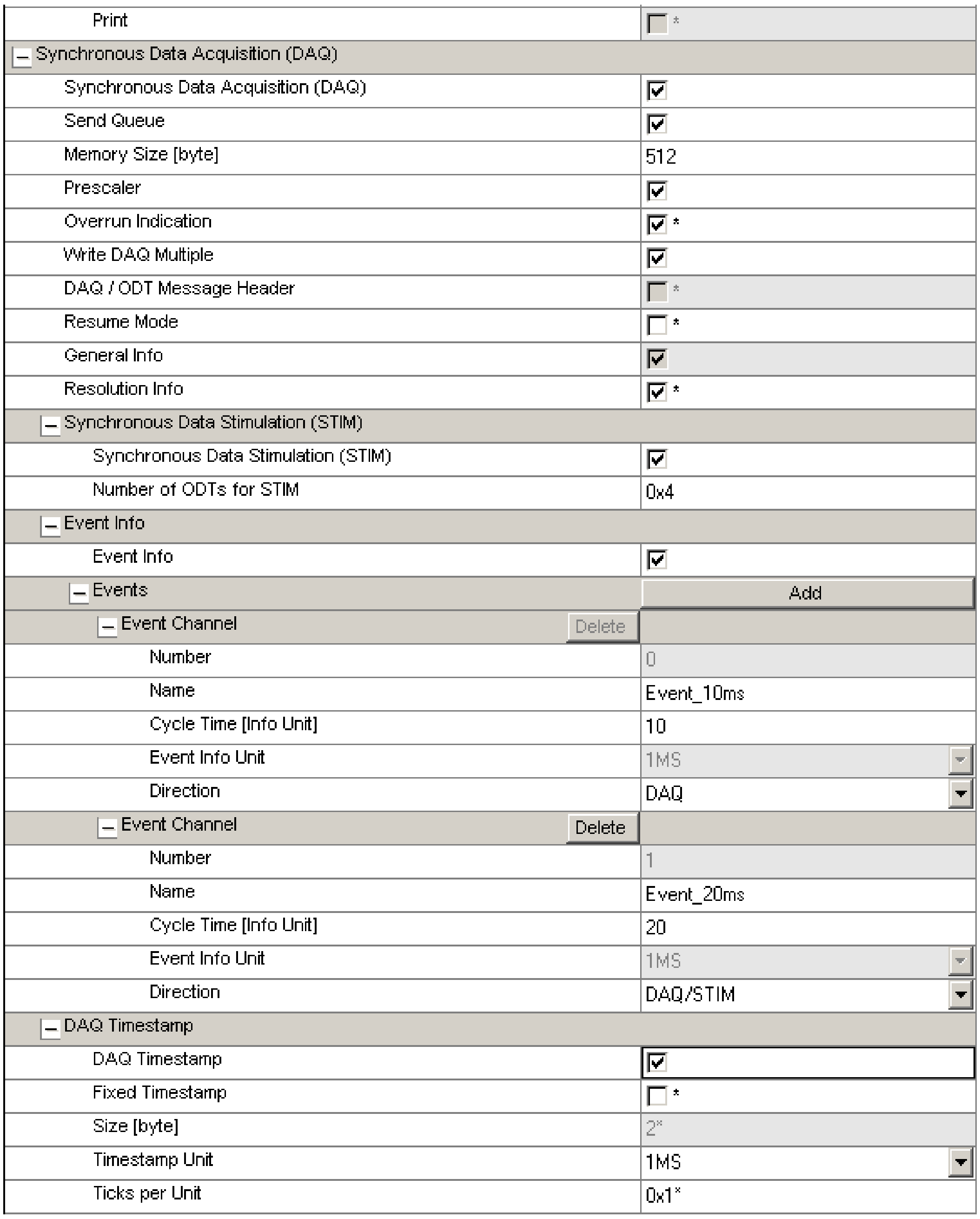
Configuration option	Description of configuration option
XCP Station Identifier	The 'XCP Station Identifier' is an ASAM-MC2 filename without path and extension that identifies the ECU's software program version. It is used for slave device identification and automatic session configuration. The Master Control System (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 to choose the wrong ECU database.
Command Parameter Check	Checks of the range and validity of Command Transfer Object (CTO) and Data Transfer Object (DTO) parameters.

Enable Calibration	The option 'Enable Calibration' unlocks the commands - DOWNLOAD - DOWNLOAD_NEXT - DOWNLOAD_MAX - SHORT_DOWNLOAD - MODIFY_BITS If this option is disabled, these commands will return an ERR_ACCESS_DENIED error and calibration of parameters will not be possible!
Event Codes	'Event Codes' are transmitted within event packets (EV) from the slave device to the master device. The transmission is not guaranteed since event packets are not acknowledged. Please refer to the XCP Protocol Layer specification for the 'Table of Event codes'.
Bootloader Download	In order to reprogram the internal flash of some microcontrollers it is necessary to use a bootloader, because code cannot be executed from flash while programming flash.
Memory Write Protection	The option 'Memory Write Protection' enables write access to memory areas. I.e. prior to carrying out write access to RAM an application callback function is called and the memory address is passed as parameter. The application has to either grant or deny the memory access.
Memory Read Protection	The option 'Memory Read Protection' enables read access to memory areas. I.e. prior to carrying out read access to RAM an application callback function is called and the memory address is passed as parameter. The application has to either grant or deny the memory access.
XCP Control	The option ‘XCP Control’ enables an API to en- or disable the XCP module (s. 3.20).
Get Xcp Data Pointer	The option ‘Get Xcp Data Pointer’ enables an API to retrieve the pointer to the internal data of the XCP module (s. 3.19)
Version Info API Support	The 'Version Info Api' option provides access to the version information of the XCP Protocol Layer module. Provided informations are Module identifier, Vendor identifier and vendor specific Version numbers.
Open Command Interface	The ‘Open Command Interface’ can be used to add unsupported XCP commands. A user call back is made available which must be implemented in the application.
Session Status API	This option enables the API XcpGetState which can be used to determine whether the XCP is in state XCP_CONNECTED or in state XCP_DISCONNECTED
Address Doubling	Address Doubling allows byte addressing on word addressable CPUs. For this purpose the Tool must double all addresses. If the Tool does not support this automatically all addresses in the a2l file must be doubled manually.

User Config File	The configuration file `xcp_cfg.h` is generated by GENy. If you want to overwrite settings in the generated configuration file, you can specify a path to a user defined configuration file. The user defined configuration file will be included at the end of the generated file. Therefore definitions in the user defined configuration file can overwrite definitions in the generated configuration file.
EEPROM Access	
Read Access	The option 'Read Access' allows read access to EEPROM. The routines for accessing the EEPROM have to be implemented in the application.
Write Access	The option 'Write Access' allows write access to EEPROM. The routines for accessing the EEPROM have to be implemented in the application.
Service Request Message	
Service Request Message	'Service Request Messages' are always transmitted within service request packets (SERV) by the slave device, in order to request the master device to perform some action. The transmission is not guaranteed since service request packets are not acknowledged by the master device. Please also refer to the XCP Protocol Layer specification for the 'Table of service request codes
Print	The function `XcpPrint(..)` can be used for the transmission of service request packets that contain text.

Table 7-1 Component configuration – General settings

7.1.2.2 Synchronous Data Acquisition



Print	☐ *
⊟ Synchronous Data Acquisition (DAQ)	
Synchronous Data Acquisition (DAQ)	☑
Send Queue	☑
Memory Size [byte]	512
Prescaler	☑
Overrun Indication	☑ *
Write DAQ Multiple	☑
DAQ / ODT Message Header	☐ *
Resume Mode	☐ *
General Info	☑
Resolution Info	☑ *
⊟ Synchronous Data Stimulation (STIM)	
Synchronous Data Stimulation (STIM)	☑
Number of ODTs for STIM	0x4
⊟ Event Info	
Event Info	☑
⊟ Events	Add
⊟ Event Channel Delete	
Number	0
Name	Event_10ms
Cycle Time [Info Unit]	10
Event Info Unit	1MS
Direction	DAQ
⊟ Event Channel Delete	
Number	1
Name	Event_20ms
Cycle Time [Info Unit]	20
Event Info Unit	1MS
Direction	DAQ/STIM
⊟ DAQ Timestamp	
DAQ Timestamp	☑
Fixed Timestamp	☐ *
Size [byte]	2*
Timestamp Unit	1MS
Ticks per Unit	0x1*

Figure 7-3 Component configuration – Synchronous Data Acquisition

Configuration option	Description of configuration option
Synchronous Data Acquisition (DAQ)	Data elements located in the slave's memory are transmitted in Data Transfer Objects (DTOs) from slave to master (DAQ) and from master to slave (STIM). The Object Description Table (ODT) describes the mapping between the synchronous data transfer objects and the slave's memory.

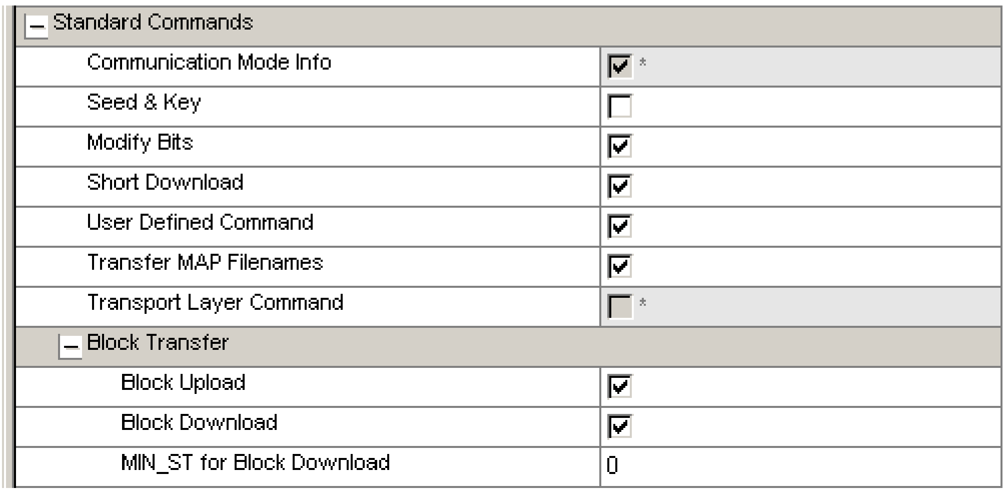
Send Queue	The ‘Send Queue’ should be enabled if more than one ODT (Object Description Table) is used and if the Transport Layer does not support data queuing or data buffering. It has to be enabled if the Vector XCP Transport Layer for CAN is enabled.
Memory Size [byte]	A memory area has to be reserved for the dynamic allocation of DAQ and ODT (Object Description Table) lists and for the transmit queue.
Prescaler	If the option 'Prescaler' is enabled all DAQ lists support the prescaler for reducing the transmission period.
Overrun Indication	Overrun situations are indicated to the Master Control System. An overrun situation is e.g. an overflow of the transmit queue.
Write DAQ Multiple	This command allows downloading multiple DAQ list entries in one CMD frame. This option only works if: 1. MAX_CTO is at least 16 bytes big 2. This feature is enabled in CANape (Extended driver settings)
DAQ / ODT Message Header	If the option 'DAQ/ODT message header' is enabled the 2 byte DAQ/ODT XCP Packet Identification is used: Relative ODT number (1 byte), absolute DAQ list number (1 byte). If the option 'DAQ/ODT message header' is disabled a 1 byte Packet Identification (PID) is used: Absolute ODT number. Attention: The 'DAQ/ODT Message Header' must not be enabled if the XCP Transport Layer for CAN or FlexRay is enabled.
Resume Mode	The option 'Resume Mode' or often also called 'Cold Start Measurement' allows automatic data transfer (DAQ, STIM) directly after power-up of the slave without prior connection to the master calibration system. Also prior set calibration data can be restored.
General Info	The option 'General Info' enables the XCP command GET_DAQ_PROCESSOR_INFO, which provides general information on DAQ lists.
Resolution Info	The option 'Resolution Info' enables the command GET_DAQ_RESOLUTION_INFO, which provides information on the resolution of DAQ lists.
Synchronous Data Stimulation (STIM)	
Synchronous Data Stimulation (STIM)	'Synchronous Data Stimulation (STIM)' is the inverse mode of 'Synchronous Data Acquisition (DAQ)'. Data elements located in the slave's memory are transmitted in Data Transfer Objects from the master device to the slave device. These data elements are written to RAM upon XCP events.
Number of ODTs for STIM	The maximum number of Object Descriptor Tables (ODTs) for Synchronous Data Stimulation (STIM) has to be configured.
Event Info	

Event Info	The option 'Event Info' enables the XCP command GET_DAQ_EVENT_INFO, which provides the following information about event channels: > Number of event channel > Name of event channel > Measurement cycle time of event channel > Direction of event channel: DAQ, STIM, DAQ&STIM
Events	The information about event channels, which is transferred from the slave device to the master device, can be configured. Attention: The number of the event channels has to be dense and zero-based
Event Channel	For each 'Event Channel' information can be configured. This information is transferred from the slave device to the master device.
Number	The event channel numbers have to be dense and zero-based. Therefore this number can not be entered manually. The event channel number is passed as a argument to the function `XcpEvent(..)`.
Name	The name of the event channel is used to identify an event within the master control system.
Cycle Time [Event Info Unit]	The 'Cycle Time' of the event channel is transferred to the master control system and used to set up the master control system.
Event Info Unit	Select the resolution of the time stamp ticks.
Direction	The following data acquisition 'Directions' of event channels are possible: > DAQ: send cyclic data transfer packets from the slave device to the master control system > STIM: send cyclic data transfer packets from the master control system to the slave device > DAQ/STIM: both directions are possible, but not simultaneously
DAQ Timestamp	
DAQ Timestamp	Timestamps can be attached to Data Transfer Object (DTO) Packets, to avoid measurement errors due to bus latency. The timestamp unit and ticks per unit have to be defined if timestamps are used.
Fixed Timestamp	If the 'Fixed Timestamp' option is selected the slave always sends Data Transfer Object (DTO) Packets in time stamped mode. Otherwise timestamps are dynamically and individually enabled for each DAQ list.
Size [byte]	Size of Timestamp. Possible timestamp sizes are 1Byte, 2Bytes and 4Bytes.
Timestamp Unit	Select the resolution of the time stamp ticks.

Ticks per Unit	The timestamp will increment per unit by the value specified here and wrap around if an overflow occurs.

Table 7-2 Component configuration – Synchronous Data Acquisition

7.1.2.3 Standard Commands



Standard Commands	
Communication Mode Info	✓ *
Seed & Key	☐
Modify Bits	✓
Short Download	✓
User Defined Command	✓
Transfer MAP Filenames	✓
Transport Layer Command	☐ *
Block Transfer	
Block Upload	✓
Block Download	✓
MIN_ST for Block Download	0

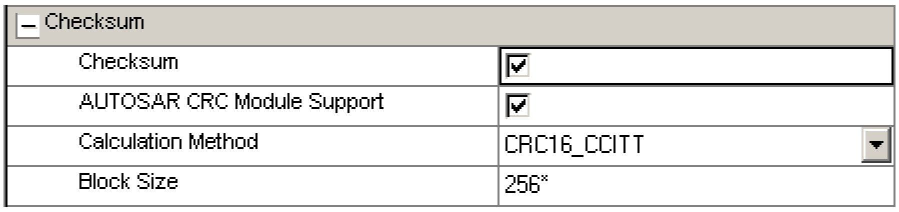
Figure 7-4 Component configuration – Standard Commands

Configuration option	Description of configuration option
Communication Mode Info	The XCP command 'GET_COMM_MODE_INFO' returns optional information on different Communication Modes supported by the slave and also the version number of the Protocol Layer implementation. If the master block mode is supported, also the maximum allowed block size and the minimum separation time are returned. The XCP Protocol Layer supports the Standard Communication model and also the Master Block Transfer Mode and the Slave Block Transfer Mode.
Seed & Key	Resources within the slave device can be protected by a 'Seed & Key' mechanism. The following resources can be protected: > Synchronous data acquisition (DAQ) > Synchronous data stimulation (STIM) > Online calibration (CAL) > Programming (PGM)
Modify Bits	This command can be en- or disabled.
Short Download	This command can be en- or disabled. For bus systems with maximum data length less equal eight (e.g. CAN, LIN) this command make no sense as no data can be transported in addition to the address information.

User Defined Command	The 'User Defined Command' is optional and can be implemented within the application. However it must not be used to implement functionalities done by other services. The application callback function ApplXcpUserService() is provided to perform application specific actions.
Transfer MAP Filenames	Slave identification via MAP filenames is one kind of user defined slave identification. If the option 'Transfer MAP Filenames' is enabled the slave device identification is done with the help of MAP files and it is possible to use the MAP filenames for automatic session configuration in the MCD tool. Therefore the MAP filenames are transferred from the XCP slave device to the MCD-tool in the initialization phase.
Transport Layer Command	The option 'Transport Layer Command' has to be enabled if transport layer specific commands are used and supported by the transport layer component.
Block Transfer	
Block Upload	The Slave Block Transfer Mode speeds up memory upload by transmitting an entire block of continuous response packets. There is only a response packet before and after transmission of the entire block. There are no limitations allowed for the master device. The slave returns whether it supports Slave Block Transfer Model in the response of the request CONNECT.
Block Download	The Master Block Transfer Mode speeds up memory download by transmitting an entire block of continuous request packets. There is only one response packet after transmission of the entire block. The XCP Master has to meet the slave's limitations of the maximum block size and the minimum separation time. These communication parameters are responded within the response to GET_COMM_MODE_INFO.
MIN_ST for Block Download	MIN_ST indicates the required minimum separation time between the packets of a block transfer from the master device to the slave device in units of 100 microseconds. The value given in GENy is transmitted within the response to the command GET_COMM_MODE_INFO.

Table 7-3 Component configuration – Standard Commands

7.1.2.4 Checksum



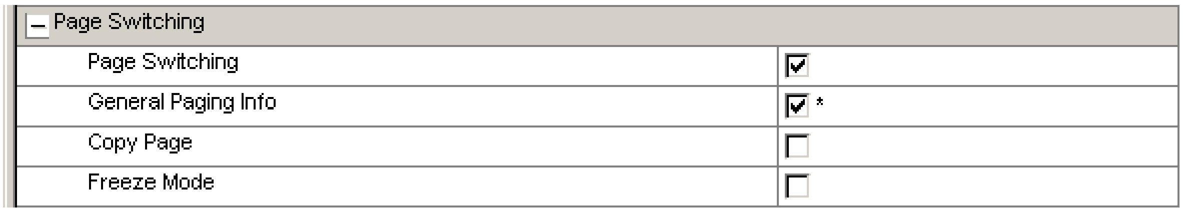
Checksum	
Checksum	✓
AUTOSAR CRC Module Support	✓
Calculation Method	CRC16_CCITT
Block Size	256*

Figure 7-5 Component configuration – Checksum

Configuration option	Description of configuration option
Checksum	The XCP command BUILD_CHECKSUM returns a checksum that is calculated over the memory block defined by the Memory Transfer Address (MTA) and block size. The MTA will be post-incremented by the block size. The checksum type (size of the checksum) and the calculation method can be configured.
AUTOSAR CRC Module Support	If ‘AUTOSAR CRC Module Support’ is enabled only the following checksum calculation methods are available: > CRC16_CCITT: CRC16 CCITT algorithm > CRC32: CRC32 algorithm The CRC32 algorithm is only supported if the AUTOSAR CRC Module is used.
Calculation Method	The following checksum calculation methods are supported: > ADD_11: add a BYTE into a BYTE checksum > ADD_12: add a BYTE into a WORD checksum > ADD_14: add a BYTE into a DWORD checksum > ADD_22: add a WORD into a WORD checksum > ADD_24: add a WORD into a DWORD checksum > ADD_44: add a DWORD into a DWORD checksum > CRC16_CCITT: CRC16 CCITT algorithm > CRC32: CRC32 algorithm The CRC32 algorithm is only supported if the AUTOSAR CRC Module is used. All checksum calculation algorithms except of the CRC algorithms ignore overflows. The block size has to be a multiple of the size of the type that is added.
Block Size	Please refer to the help of 'Checksum'.

Table 7-4 Component configuration – Checksum

7.1.2.5 Page Switching



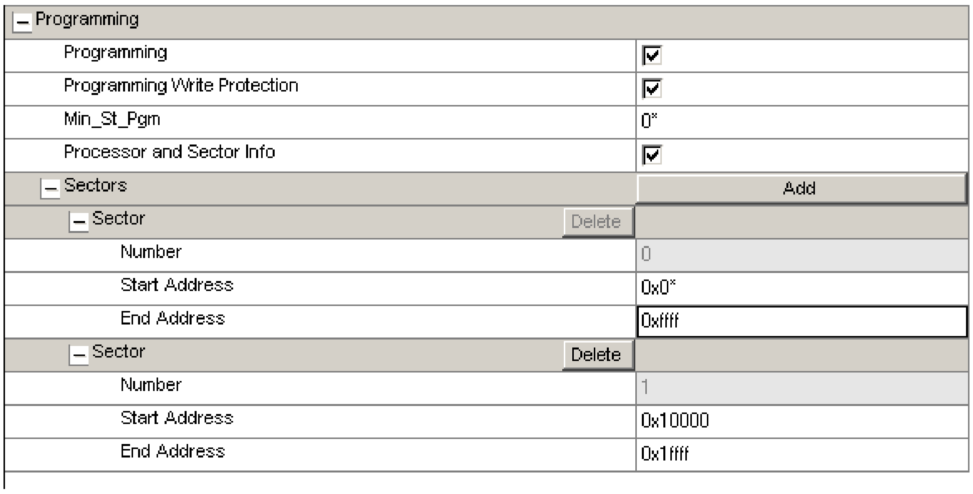
Page Switching	
Page Switching	✓
General Paging Info	✓ *
Copy Page	☐
Freeze Mode	☐

Figure 7-6 Component configuration – Page Switching

Configuration option	Description of configuration option
Page Switching	If calibration page switching (PAG) is enabled the access mode calibration data segments can be set. Calibration data segments and their pages are specified by logical numbers.
General Paging Info	If 'General Paging Info' is enabled the XCP command 'GET_PAG_PROCESSOR_INFO' returns general information on paging. The following information is transferred from the slave device to the master device: > The total number of segments > Whether the freeze mode is supported Specific information for segments or pages is so far not supported.
Copy Page	If more than one calibration page is defined, the slave can copy a calibration page into another. In principle any page of any segment can be copied to any page of any segment. However, restrictions might be possible.
Freeze Mode	If enabled the commands SET_SEGMENT_MODE and GET_SEGMENT_MODE are enabled and forwarded to the application. Enabling this feature also set the Freeze Mode Supported bit in General Paging Info

Table 7-5 Component configuration – Page Switching

7.1.2.6 Programming



– Programming	
Programming	✓
Programming Write Protection	✓
Min_St_Pgm	0*
Processor and Sector Info	✓
– Sectors	Add
– Sector Delete	
Number	0
Start Address	0x0*
End Address	0xffff
– Sector Delete	
Number	1
Start Address	0x10000
End Address	0x1ffff

Figure 7-7 Component configuration – Programming

Configuration option	Description of configuration option
Programming	The option 'Programming' enables the programming of non-volatile memory. If the internal flash of the microcontroller cannot be programmed while execution of code from the flash, the 'bootloader download' functionality has to be used instead.
Programming Write Protection	The option 'Programming Write Protection' enables the programming write protection of non-volatile memory. I.e. prior to carrying out write access to non-volatile memory an application callback function (see 6.5.10) is called and the memory address is passed as parameter. The application has to either grant or deny the memory access.
Min_St_Pgm	This parameter defines the delay the Master should insert between two consecutive PROGRAM_NEXT commands. This parameter is only relevant if Block Mode is used.
Processor and Sector Info	The option 'Processor and Sector Info' enables the commands: > GET_PGM_PROCESSOR_INFO Transfers the general properties for programming and the total number of available sectors from the slave device to the master device. > GET_SECTOR_INFO Transfers information on a specific sector from the slave device to the master device.
Sectors	The information for sectors, which is transferred from the slave device to the master device, can be configured. Attention: The sector number has to be dense and zero-based!
Sector	For each 'Sector' information can be configured. This information is transferred from the slave device to the master device.
Number	The sector numbers have to be dense and zero-based. Therefore this number can not be entered manually.
Start Address	The 'Start Address' of each sector is individually configured in the slave device and transferred to the master device.
End Address	The 'End Address' of each sector is individually configured in the slave device and transferred to the master device.

Table 7-6 Component configuration – Programming

7.1.2.7 Generated a2l files

GENy 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)

Example Master.a2l:

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

7.2 Configuration without Generation Tool

The configuration of the configuration switches and constants is done in the file `xcp_cfg.h`. An example that contains the default configuration of XCP Basic is distributed together with XCP Basic. It is recommended to use this example as a template for the individual configuration.

7.2.1 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 that common of XCP Basic and XCP Professional. The default values are bold.

Configuration switches	Value	Description
`XCP_xxx_DAQ`	`ENABLE`, **`DISABLE`**	Enables/disables synchronous data acquisition.
`XCP_xxx_DAQ_PRESCALER`	`ENABLE`, **`DISABLE`**	Enables/disables the DAQ prescaler.
`XCP_xxx_DAQ_OVERRUN_INDICATION`	`ENABLE`, **`DISABLE`**	Enables/disables the DAQ overrun detection.

`XCP_xxx_DAQ_HDR_ODT_DAQ`[3]	`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
`XCP_xxx_DAQ_PROCESSOR_INFO`	`ENABLE`, **`DISABLE`**	Plug & play mechanism for the data acquisition processor.
`XCP_xxx_DAQ_RESOLUTION_INFO`	`ENABLE`, **`DISABLE`**	Plug & play mechanism for the data acquisition resolution.
`XCP_xxx_DAQ_EVENT_INFO`	`ENABLE`, **`DISABLE`**	Plug & play mechanism for the event definitions.
`XCP_xxx_DAQ_TIMESTAMP`	`ENABLE`, **`DISABLE`**	DAQ timestamps
`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_CRC16CCITT_REFLECTED`	`ENABLE`, **`DISABLE`**	Enable/disable reflected CRC16 CCITT checksum calculation algorithm. Also refer to 7.2.2.1 '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_TL_COMMAND`	`ENABLE`, **`DISABLE`**	Transport Layer command
`XCP_xxx_COMM_MODE_INFO`	`ENABLE`, **`DISABLE`**	Communication mode info

[3] 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_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[4]	ENABLE, **DISABLE**	Supports the usage of dual port RAM
XCP_xxx_GET_CONNECTION_STATE	ENABLE, **DISABLE**	Get connection state of XCP

The following table contains an additional list of all configuration switches, used to control the functional units that are only available in XCP Professional. The default values are bold.

Configuration switches	Value	Description
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

[4] Not supported by XCP Professional

`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
`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 API for en- / disable XCP module

The following table contains an additional list of all configuration switches, used to control the functional units that are only available in XCP basic. The default values are bold.

Configuration switches	Value	Description
`XCP_ENABLE_TESTMODE`[5]	ENABLE, **DISABLE**	Test mode that allows the output of debugging information. Not included in XCP Professional due to multiple MISRA rule violations!

7.2.2 Configuration of Constant Definitions

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

Constant definitions	Range	Default	Description
`kXcpMaxCTO`	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.
`kXcpMaxDTO`	8..255[6]	**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.

[5] Not supported by XCP Professional
[6] Implementation specific range. The range is 8..0xFFFF according to XCP specification [I], [II].

`kXcpDaqMemSize`	`0.. 0xFFFF`	**256**	Define the amount of memory used for the DAQ lists and buffers. Also refer to chapter 8 (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`[7]	-	Number of available events in the slave (part of event channel plug & play mechanism) Also refer to chapter 7.2.6.
`kXcpStimOdtCount`	`0..0xC0`	**0xC0**	Maximum number of ODTs that may be used for Synchronous Data Stimulation.
`kXcpChecksumMethod`	-	-	Checksum calculation method. Refer to chapter 7.2.2.1 'Table of Checksum Calculation Methods' for valid values.
`kXcpChecksumBlockSize`	`1 .. 0xFFFF`	**256**	Each call of `XcpBackground` 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 7.2.8
`kXcpMaxSegment`	`1`	**1**	Number of memory segments Also refer to chapter 7.2.9.
`kXcpMaxPages`	`1..2`	**2**	Number of pages Also refer to chapter 7.2.9.

7.2.2.1 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
`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

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

`XCP_CHECKSUM_TYPE_CRC16CCITT`	CRC16 CCITT checksum calculation algorithm Both the standard and the reflected algorithm are supported. Please refer to chapter 10.8 ‘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.

7.2.3 Definition of Memory Qualifiers

The definition of the memory qualifiers has to be customized depending on the controller and memory model.

Type	Default	Description
`vuint8`	`unsigned char`	Unsigned 8-bit identifier
`vuint16`	`unsigned short`	Unsigned 16-bit identifier
`vuint32`	`unsigned long`	Unsigned 32-bit identifier
`V_MEMROM0`		Addition qualifier to access data in ROM
`MEMORY_ROM_NEAR`	`const`	Fast data access in ROM
`MEMORY_ROM`	`const`	Default according to memory model in ROM
`MEMORY_ROM_FAR`	`const`	Slow addressing mode in ROM
`MEMORY_NEAR`		Short addressed RAM
`MEMORY_NORMAL`		Default addressed RAM
`MEMORY_FAR`		Far addressed RAM
`P_MEM_ROM`		Pointer to ROM
`P_MEM_RAM`		Pointer to RAM

7.2.4 Configuration of the CPU Type

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

Configuration switches	Value	Description
`C_CPUTYPE_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_DISBLE_UNALIGNED_MEM_ACCESS` is defined WORDs are located on WORD aligned and DWORD are located on DWORD aligned addresses.

7.2.5 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.

7.2.5.1 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:

```
V_MEMROM0 vuint8 MEMORY_ROM kXcpStationId[] = "station ID"
```

The range of `kXcpStationIdLength` is `0..0xFF`.

7.2.5.2 Automatic Session Configuration with MAP Filenames

The automatic session configuration by transferring MAP filenames is enabled with the switch: `XCP_ENABLE_VECTOR_MAPNAMES`

7.2.6 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 top be defined as described below:

Constant	Range	Description
`kXcpMaxEvent`	`0..0xFF`[8]	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: - `kXcpEventDirectionDaq` For XCP Professional valid values are: - `kXcpEventDirectionDaq` - `kXcpEventDirectionStim` - `kXcpEventDirectionDaqStim`

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

Example

```
#define XCP_ENABLE_DAQ_EVENT_INFO
#define kXcpMaxEvent 3

V_MEMROM0  static  vuint8  MEMORY_ROM  kXcpEventName_0[]  =
"10ms";
V_MEMROM0  static  vuint8  MEMORY_ROM  kXcpEventName_1[]  =
"100ms DAQ";
V_MEMROM0  static  vuint8  MEMORY_ROM  kXcpEventName_2[]  =
"100ms STIM";
V_MEMROM0 MEMORY_ROM vuint8* MEMORY_ROM kXcpEventName[] =
{
  &kXcpEventName_0[0],
  &kXcpEventName_1[0],
  &kXcpEventName_2[0]
};

V_MEMROM0 vuint8 MEMORY_ROM kXcpEventNameLength[] =
{
  4,
  9,
  10
};

V_MEMROM0 vuint8 MEMORY_ROM kXcpEventCycle[] =
{
  10,
  100,
  100
};

V_MEMROM0 vuint8 MEMORY_ROM kXcpEventDirection[] =
{
  kXcpEventDirectionDaq,
  kXcpEventDirectionDaq,
```

	`  kXcpEventDirectionStim`
	`};`

7.2.7 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
kXcpDaqTimestampSize	DAQ_TIMESTAMP_BYTE, DAQ_TIMESTAMP_WORD, DAQ_TIMESTAMP_DWORD	This parameter defines the size of timestamps. It can either be 1 byte, 2 bytes or 4 bytes.
XcpDaqTimestampType	vuint8, vuint16 or vuint32	Type of the timestamp depends on the parameter kXcpDaqTimestampSize.
kXcpDaqTimestampUnit	DAQ_TIMESTAMP_UNIT_1NS 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	Unit of the timestamp (1 ns, 10 ns .. 1 s)
kXcpDaqTimestampTicksPerUnit	0..0xFFFF	Time stamp ticks per unit

7.2.8 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
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

V_MEMROM0 vuint32 MEMORY_ROM kXcpProgramSectorStart [] =
{
   (vuint32)0x000000u,
   (vuint32)0x010000u,
};
V_MEMROM0 vuint32 MEMORY_ROM kXcpProgramSectorEnd [] =
{
   (vuint32)0x00FFFFu,
   (vuint32)0x01FFFFu,
};
```

7.2.9 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 top be defined as described below:

Constant	Range	Description
`kXcpMaxSegment`	`0x01`	Number of memory segments
`kXcpMaxPages`	`0x01..0x02`	Number of pages

8 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.

9 Limitations

9.1 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 and events 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

Planned:

- User defined checksum calculations
- CRC16 and CRC32

9.2 Limitations of XCP Basic

The XCP Protocol Layer is available in two variants:

- XCP Professional Version
- XCP Basic Version

The XCP Professional Version is the ‘full version’, which is also supported by the Vector generation tool GENy. The XCP Basic Version is a subset of the ‘full version’, which is distributed freely via the internet and which has to be configured manually.

The XCP features that are available by the XCP Professional version but not by the XCP Basic version are listed below:

- Stimulation (Bypassing)
- Bit stimulation[9]
- Atomic bit manipulation
- SHORT_DOWNLOAD
- FLASH and EEPROM Programming
- The block transfer communication mode
- Resume mode
- The transmission of service request packets
- Memory write protection
- Memory read protection
- Programming write protection
- Support of AUTOSAR CRC module
- Access to internal data pointer
- XCP deactivation
- Open Command Interface
- Transport Layer Commands
- Configurable timestamp size
- Disable Calibration

[9] Not yet supported by XCP Professional

9.3 Limitations Regarding Platforms, Compilers and Memory Models

Even though XCP Professional and XCP Basic are Protocol Layers and therefore higher software layers, they manipulate 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

10 FAQ

10.1 Connection to MCS Not Possible

FAQ
After integration of XCP on CAN or integration of XCP Basic with a proprietary CAN-Driver does the MCS (e.g. CANape) not connect with the XCP slave, even though the CAN communication is working properly.

The XCP protocol allows transmitting XCP packets with a variable data length. However many OEMs require that all CAN messages sent within their automotive networks have to have a static DLC. Therefore messages sent by the MCS with a DLC of less than 8 (e.g. CONNECT has a DLC of 2) might be discarded by the ECU’s CAN-Driver and the connection is not possible.

Check whether your MCS supports transmission with static DLC. This is supported by CANape since Version 5.5.

10.2 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.

10.3 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 vuint8 XCP_MEMORY_FAR *`
  This pointer is used to access memory for standard read and write operations

- `DAQBYTEPTR`
  `#define DAQBYTEPTR vuint8 XCP_MEMORY_FAR *`
  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.

10.4 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.
There are two options available. Both have to be configured within the user config file:

- Far access to the internal data of XCP:
  `#define FAR far`

- Disable type casts from pointers to integers :
  `#define XCP_ENABLE_NO_P2INT_CAST`

10.5 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.

10.6 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 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.

10.7 Can XCP memory be placed in far RAM?

	FAQ How can the internal XCP memory be placed in far RAM?

The current implementation does not allow the XCP memory to be easily placed in far RAM. What you can do is to compile the whole component in the respective memory model to allow far memory access.

10.8 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.

11 Bibliography

This manual refers to the following documents:

[I] XCP -Part 1 - Overview
Version 1.0 of 2003-04-08

[II] XCP -Part 2- Protocol Layer Specification
Version 1.0 of 2003-04-08

[III] XCP -Part 5- Example Communication Sequences
Version 1.0 of 2003-04-08

[IV] Technical Reference XCP on CAN Transport Layer
Version 1.4 of 2006-04-24

[V] Technical Reference XCP on FlexRay Transport Layer
Version 1.0 of 2005-12-21

[VI] Technical Reference XCP on LIN Transport Layer
Version 1.0 of 2006-05-30

[VII] AUTOSAR Specification of CRC Routines
Release 2.0.0 of 2006-04-28

12 Contact

Visit our website for more information on

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

www.vector-informatik.com