

# XCP Protocol Layer

## Technical Reference

Version 1.0

<b>Authors:</b>	Frank Triem
<b>Version:</b>	1.0
<b>Status:</b>	released (in preparation/completed/inspected/released)

## 1 History

Author	Date	Version	Remarks
Frank Triem Klaus Emmert	2005-01-17	1.0	ESCAN00009143: Initial draft Warning Text added

**Please note**

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

## Contents

<b>1</b>	<b>History .....</b>	<b>2</b>
<b>2</b>	<b>Overview .....</b>	<b>8</b>
2.1	Abbreviations and items used in this paper .....	9
2.2	Naming conventions.....	10
<b>3</b>	<b>Functional description .....</b>	<b>11</b>
3.1	Overview of the functional scope .....	11
3.2	Communication mode info .....	11
3.3	Block transfer communication model (XCP Professional only) .....	11
3.4	Slave device identification .....	11
3.4.1	XCP Station Identifier.....	11
3.4.2	Transferring of XCP MAP filenames .....	12
3.5	Seed & Key .....	12
3.6	Checksum calculation .....	13
3.7	Memory protection (XCP Professional only) .....	14
3.8	Event Codes.....	14
3.9	Service Request Messages (XCP Professional only) .....	14
3.10	User defined command .....	15
3.11	Synchronous Data Transfer .....	15
3.11.1	Synchronous data acquisition (DAQ) .....	15
3.11.2	DAQ timestamp .....	15
3.11.3	Power-up data transfer (XCP Professional only) .....	15
3.11.4	Data stimulation (STIM) (XCP Professional only) .....	16
3.11.5	Bypassing (XCP Professional only) .....	16
3.11.6	Data acquisition plug & play mechanisms.....	16
3.11.7	Event channel plug & play mechanism .....	17
3.12	The online data calibration model .....	17
3.12.1	Page switching .....	17
3.12.2	Page switching plug & play mechanism .....	17
3.12.3	Calibration data page freezing .....	17
3.12.4	Calibration data page copying.....	18
3.13	Flash Programming (XCP Professional only).....	18
3.13.1	Flash Programming by the ECU's application.....	18
3.13.1.1	Flash Programming plug & play mechanism.....	18
3.13.2	Flash Programming with a flash kernel .....	19
3.14	EEPROM access (XCP Professional only) .....	19
3.15	Parameter check .....	20

3.16	Performance optimizations.....	20
3.17	Interrupt locks.....	20
<b>4</b>	<b>Integration into the application.....</b>	<b>21</b>
4.1	Files of XCP Professional .....	21
4.2	Files of XCP Basic .....	21
4.3	Version changes .....	22
4.4	Integration of XCP into the application.....	22
4.4.1	Integration of XCP on CAN (XCP Professional only) .....	22
4.4.2	Integration with a proprietary XCP Transport Layer.....	23
4.4.3	Motorola HC12 with CAN transport layer .....	25
<b>5</b>	<b>Feature List.....</b>	<b>26</b>
<b>6</b>	<b>Description of the API.....</b>	<b>28</b>
6.1	Version of the source code .....	28
6.2	XCP Services called by the Application .....	29
6.2.1	XcpInit: Initialization of the XCP Protocol Layer.....	29
6.2.2	XcpEvent: Handling of a data acquisition event channel .....	29
6.2.3	XcpStimEventStatus: Check data stimulation events .....	30
6.2.4	XcpBackground: Background calculation of checksum .....	31
6.2.5	XcpSendEvent: Transmission of event codes.....	31
6.2.6	XcpPutchar: Put a char into a service request packet .....	32
6.2.7	XcpPrint: Transmission of a service request packet .....	32
6.2.8	XcpDisconnect: Disconnect from XCP master.....	33
6.2.9	XcpSendCrm: Transmit response or error packet.....	34
6.3	XCP Protocol Layer functions, called by the XCP Transport Layer .....	35
6.3.1	XcpCommand: Evaluation of XCP packets and command interpreter.....	35
6.3.2	XcpSendCallBack: Confirmation of the successful transmission of a XCP packet .....	36
6.4	XCP Transport Layer services called by the XCP Protocol Layer .....	37
6.4.1	ApplXcpSend: Request for the transmission of a DTO or CTO message..	37
6.4.2	ApplXcpInit: Perform XCP Transport Layer initialization .....	37
6.4.3	ApplXcpBackground: XCP Transport Layer background operations .....	38
6.4.4	ApplXcpInterruptEnable: Enable interrupts .....	38
6.4.5	ApplXcpInterruptDisable: Disable interrupts .....	39
6.5	Application services called by the XCP Protocol Layer.....	40
6.5.1	ApplXcpGetPointer: Pointer conversion.....	40
6.5.2	ApplXcpGetIdData: Get MAP filenames .....	41
6.5.3	ApplXcpGetSeed: Generate a seed .....	41
6.5.4	ApplXcpUnlock: Valid key and unlock resource .....	42

6.5.5	ApplXcpCheckReadEEPROM: Check read access from EEPROM .....	43
6.5.6	ApplXcpCheckWriteEEPROM: Check write access to the EEPROM .....	43
6.5.7	ApplXcpCheckWriteAccess: Check address for valid write access .....	44
6.5.8	ApplXcpUserService: User defined command .....	45
6.5.9	ApplXcpSendStall: Resolve a transmit stall condition .....	45
6.5.10	ApplXcpSendFlush: Flush transmit buffer .....	46
6.5.11	ApplXcpDisableNormalOperation: Disable normal operation of the ECU ..	46
6.5.12	ApplXcpStartBootLoader: Start of boot loader .....	47
6.5.13	ApplXcpReset: Perform ECU reset .....	48
6.5.14	ApplXcpFlashClear: Clear flash memory .....	48
6.5.15	ApplXcpFlashProgram: Program flash memory .....	49
6.5.16	ApplXcpDaqResume : Resume automatic data transfer .....	49
6.5.17	ApplXcpDaqResumeStore : Store DAQ lists for resume mode .....	50
6.5.18	ApplXcpDaqResumeClear : Clear stored DAQ lists .....	51
6.5.19	ApplXcpGetTimestamp: Returns the current timestamp .....	51
6.5.20	ApplXcpGetCalPage: Get calibration page .....	52
6.5.21	ApplXcpSetCalPage: Set calibration page .....	52
6.5.22	ApplXcpCopyCalPage: Copying of calibration data pages .....	53
6.6	XCP Protocol Layer functions that can be overwritten .....	55
6.6.1	XcpMemCpy: Copying of a memory range .....	55
6.6.2	XcpMemSet: Initialization of a memory range .....	55
6.6.3	XcpMemClear: Clear a memory range .....	56
6.6.4	XcpSendDto: Transmission of a data transfer object .....	57
<b>7</b>	<b>Configuration of the XCP Protocol Layer .....</b>	<b>58</b>
7.1	Compiler switches .....	58
7.2	Configuration of constant definitions .....	61
7.2.1	Table of checksum calculation methods .....	62
7.3	Definition of memory qualifiers .....	62
7.4	Configuration of the CPU type .....	62
7.5	Configuration of slave device identification .....	63
7.5.1	Identification by ASAM-MC2 filename without path and extension .....	63
7.5.2	Automatic session configuration with MAP filenames .....	63
7.6	Configuration of the event channel plug & play mechanism .....	63
7.7	Configuration of the DAQ time stamped mode .....	65
7.8	Configuration of the flash programming plug & play mechanism .....	66
7.9	Configuration of the page switching plug & play mechanism .....	66
<b>8</b>	<b>Resource Requirements .....</b>	<b>67</b>
<b>9</b>	<b>Limitations .....</b>	<b>68</b>

9.1	General limitations .....	68
9.2	Limitations of XCP Basic.....	69
<b>10</b>	<b>FAQ</b>	<b>70</b>
10.1	Connection to MCS not possible .....	70
<b>11</b>	<b>Bibliography .....</b>	<b>71</b>
<b>12</b>	<b>Contact</b>	<b>72</b>

**Illustrations**

figure 4-1      Integration of XCP on CAN into the application..... 22

figure 4-2      Integration of XCP with a proprietary XCP Transport Layer ..... 23

## 2 Overview

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

This document does not cover the XCP on CAN Transport Layer. Please refer to [IV] for further information about XCP on CAN and the integration of XCP on CAN with the Vector CANbedded software components.

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 on CAN.

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](http://www.vector-informatik.de) in a functional restricted form.



## 2.1 Abbreviations and items used in this paper

Abbreviations	Complete expression
<b>A2L</b>	File Extension for an <b>ASAM 2MC Language File</b>
<b>AML</b>	<b>ASAM 2 Meta Language</b>
<b>API</b>	<b>Application Programming Interface</b>
<b>ASAM</b>	<b>Association for Standardization of Automation and Measuring Systems</b>
<b>BYP</b>	<b>BYPassing</b>
<b>CAN</b>	<b>Controller Area Network</b>
<b>CAL</b>	<b>CALibration</b>
<b>CANape</b>	Calibration and Measurement Data Acquisition for Electronic Control Systems
<b>CMD</b>	<b>Command</b>
<b>CTO</b>	<b>Command Transfer Object</b>
<b>DAQ</b>	Synchronous <b>Data Acquisition</b>
<b>DLC</b>	<b>Data Length Code</b> ( Number of data bytes of a CAN message )
<b>DLL</b>	<b>Data link layer</b>
<b>DTO</b>	<b>Data Transfer Object</b>
<b>ECU</b>	<b>Electronic Control Unit</b>
<b>ERR</b>	<b>Error Packet</b>
<b>EV</b>	<b>Event packet</b>
<b>ID</b>	<b>Identifier</b> (of a CAN message)
<b>Identifier</b>	Identifies a CAN message
<b>ISR</b>	Interrupt <b>Service Routine</b>
<b>MCS</b>	<b>Master Calibration System</b>
<b>Message</b>	One or more signals are assigned to each message.
<b>ODT</b>	<b>Object Descriptor Table</b>
<b>OEM</b>	<b>Original equipment manufacturer</b> (vehicle manufacturer)
<b>PAG</b>	<b>PAGing</b>
<b>PID</b>	<b>Packet Identifier</b>
<b>PGM</b>	<b>Programming</b>
<b>RAM</b>	<b>Random Access Memory</b>
<b>RES</b>	Command <b>Response Packet</b>
<b>ROM</b>	<b>Read Only Memory</b>
<b>SERV</b>	<b>Service Request Packet</b>
<b>STIM</b>	<b>Stimulation</b>
<b>TCP/IP</b>	<b>Transfer Control Protocol / Internet Protocol</b>
<b>UDP/IP</b>	<b>Unified Data Protocol / Internet Protocol</b>
<b>USB</b>	<b>Universal Serial Bus</b>
<b>XCP</b>	<b>Universal Measurement and Calibration Protocol</b>

## 2.2 Naming conventions

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

## 3 Functional description

### 3.1 Overview of the functional scope

The Universal Measurement and Calibration Protocol (XCP) was 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 an request without anymore requests in between.

Refer to chapter 7.1 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 7.5.1 (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 return a pointer to a pointer of MAP file names.

Refer to chapter 7.5.2 (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 7.1.

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.

#### Annotation for the usage of CANape

The calculation of the key is done in a DLL named SEEDKEY1.DLL, which is developed by the ECU manufacturer and which must be located in the EXEC directory of CANape. CANape can access the ECU only if the ECU accepts the key. If the key is not valid, the ECU is locked.

#### Example Implementation for SEEDKEY1.DLL

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

#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 add algorithms and a CRC16CCITT checksum calculation algorithm. Also refer to '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)

A write access of specific RAM areas can be checked with the function

```
vuint8 ApplXcpCheckWriteAccess( MTABYTEPTR addr, vuint8 size ) (6.5.7)
```

if the switch `XCP_ENABLE_WRITE_PROTECTION` is set. It should only be used, if write protection of memory areas is required.

### 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_ENBALE_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 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 7.1 for the configuration of the service request message.

### 3.10 User defined command

The XCP Protocol allows to have a user defined command with an application specific functionality. The user defined command is enabled by setting `XCP_ENABLE_USER_COMMAND` and upon reception is the callback function

```
vuint8 ApplXcpUserService (MEMORY_ROM BYTEPTR pCmd) (6.5.8)
```

called by the XCP command processor.

### 3.11 Synchronous Data Transfer

#### 3.11.1 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.

#### Annotation for the usage of CANape

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

#### 3.11.2 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 7.7 (Configuration of the DAQ time stamped mode).

#### 3.11.3 Power-up data transfer (XCP Professional only)

Power-up data transfer (also called resume mode) allows to automatically transfer data (DAQ, STIM) directly after power-up of the slave. 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 ( tXcpData * xcpData ) (6.5.16)
```

```
vuint8 ApplXcpDaqResumeStore ( MEMORY_ROM tXcpData * xcpData ) (6.5.17)
```

```
vuint8 ApplXcpDaqResumeClear ( void ) (6.5.18)
```

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

#### Annotation for the usage of CANape

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

### 3.11.4 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.

### 3.11.5 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.

### 3.11.6 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

### 3.11.7 Event channel plug & play mechanism

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

Please refer to chapter 7.6 (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.

## 3.12 The online data calibration model

### 3.12.1 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) (6.5.20)
```

```
vuint8 ApplXcpSetCalPage (vuint8 segment, vuint8 page, vuint8 mode) (6.5.21)
```

#### 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.

### 3.12.2 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
- segment information
- information about the pages

Also refer to chapter 7.9 (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`.

### 3.12.3 Calibration data page freezing

Calibration data page freezing 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.

### 3.12.4 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 application callback function

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

 (6.5.22)

has to be provided by the application.

### 3.13 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.

#### 3.13.1 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.

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

```
vuint8 ApplXcpFlashClear( MTABYTEPTR a, vuint32 size )
```

 (6.5.14)

```
vuint8 ApplXcpFlashProgram( MEMORY_ROM BYTEPTR data,
                           MTABYTEPTR a, vuint8 size )
```

 (6.5.15)

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

#### Annotation for the usage of CANape

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

#### 3.13.1.1 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 7.8 (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`.

### 3.13.2 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. Event 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 ) (6.5.11)
```

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 an proprietary flash kernel.

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

### 3.14 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 the following callback function

```
vuint8 ApplXcpCheckReadEEPROM
( MTABYTEPTR addr, vuint8 size, BYTEPTR data ) (6.5.5)
```

checks whether EEPROM or RAM is accessed. 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_NEXT` can be used. The switch `XCP_ENABLE_WRITE_EEPROM` allows the EEPROM access for these commands.

Before writing to an address the following callback function

```

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

```

(6.5.6)

checks whether EEPROM or RAM is accessed.

### 3.15 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.

### 3.16 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 as far as possible DMA access 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.

### 3.17 Interrupt locks

The functions `XcpEvent`, `XcpSendCallBack`, `XcpBackground` and `XcpCommand` are not reentrant. If one of these function 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!





## 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 <b>must not</b> be changed by the application!	
XcpProf.h	API of XCP Professional. This file <b>must not</b> be changed by the application!	
_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 <b>must not</b> 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 XCP 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	Configuration file for the XCP Protocol Layer.	
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 stack 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 <b>must not</b> be changed by the application!	
xcpBasic.h	API of XCP Basic. This file <b>must not</b> 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:

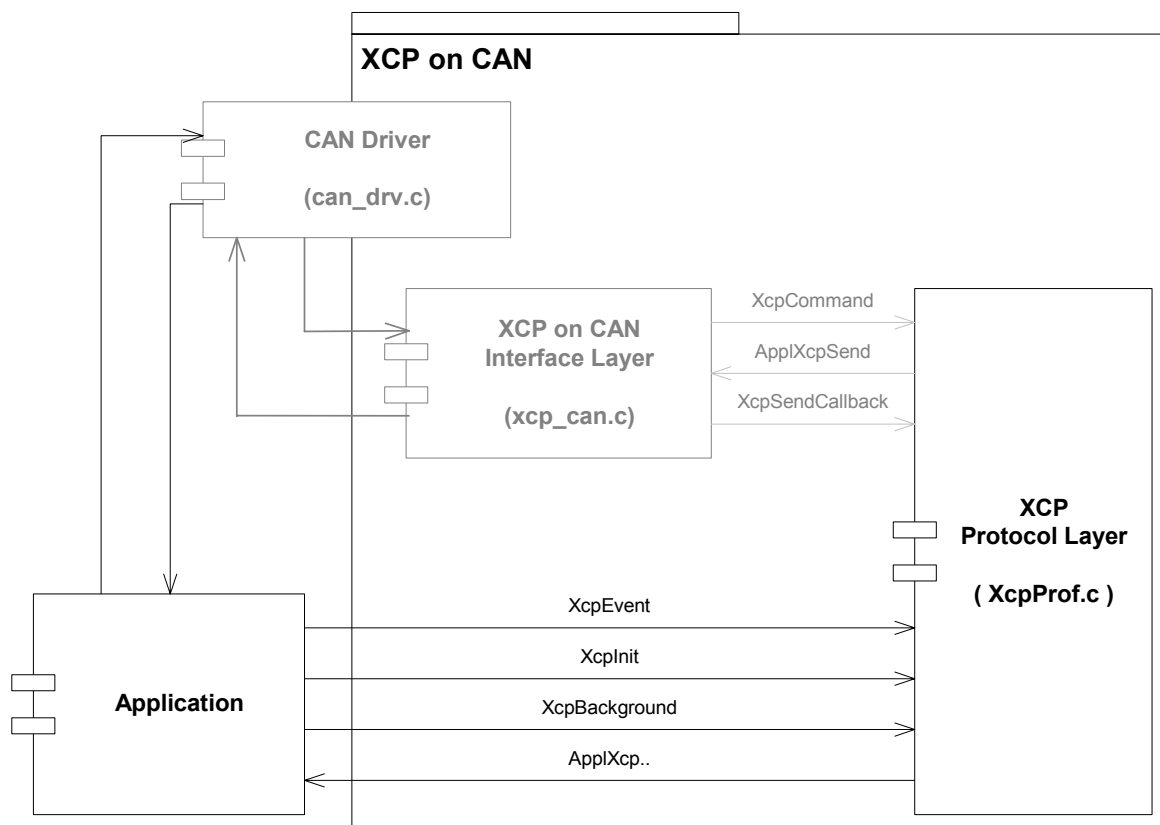


figure 4-1 Integration of XCP on CAN into the application



#### 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

	<p>XCP on CAN services or provide services that XCP on CAN uses.</p> <ol style="list-style-type: none"> <li>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.</li> <li>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 <code>XcpInit</code>.</li> <li>5. If required call the background function <code>XcpBackground</code> cyclically.</li> <li>6. Integrate the desired XCP on CAN services into your application. Call especially the function <code>XcpEvent(channel)</code> cyclic with the appropriate cycle time and channel number.</li> </ol> <p>The XCP on CAN sources must not be changed for the integration into the application.</p>
--	--

#### 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, 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 include the XCP Transport Layer for CAN.

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

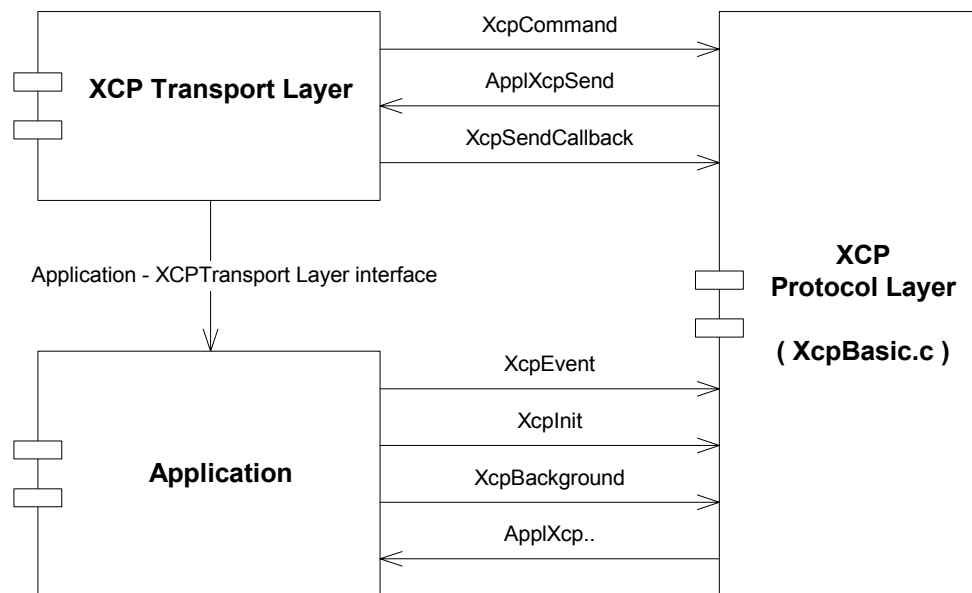


figure 4-2 Integration of XCP with a proprietary XCP Transport Layer

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 Integration of XCP with a proprietary XCP Transport Layer" 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);
    return 1;
}

/* 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
<b>Initialization</b>		
Initialization	Prof, Basic	XcpInit ApplXcpInit
<b>Task</b>		
Background task	Prof, Basic	ApplXcpBackground XcpBackground
<b>XCP Command Processor</b>		
Command Processor	Prof, Basic	XcpCommand
Transmission and Confirmation of packets	Prof, Basic	ApplXcpSend XcpSendCallback
Transmission of Response packets	Prof, Basic	XcpSendCrm
Transmission of packets	Prof, Basic	AppXcpSendStall ApplXcpSendFlush
<b>XCP Commands</b>		
Get MAP filenames	Prof, Basic	ApplXcpGetIdData
Seed & Key	Prof, Basic	ApplXcpGetSeed ApplXcpUnlock
User command	Prof, Basic	ApplXcpUserService
<b>Data Acquisition (DAQ)</b>		
Synchronous Data Acquisition and Stimulation	Prof, Basic	XcpEvent
DAQ Timestamp	Prof, Basic	ApplXcpGetTimestamp
Resume Mode	Prof	ApplXcpDaqResume ApplXcpDaqResumeStore ApplXcpDaqResumeClear
<b>Online Data Calibration</b>		
Calibration page switching	Prof, Basic	ApplXcpGetCalPage ApplXcpSetCalPage
Copy calibration page	Prof, Basic	ApplXcpCopyCalPage
<b>Boot loader Download</b>		
Disable normal operation of ECU	Prof	ApplXcpDisableNormalOperation
Start of the boot loader	Prof	ApplXcpStartBootLoader
<b>Flash Programming</b>		
Reset of ECU	Prof	ApplXcpReset
Clear flash memory	Prof	ApplXcpFlashClear

Program flash memory	Prof	ApplXcpFlashProgram
<b>Special Features</b>		
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
Overwrite able macros	Prof, Basic	XcpMemCpy XcpMemSet XcpMemClear XcpSendDto

## 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.

## 6.2 XCP Services called by the Application

The following XCP services that are called by the application must not be called within interrupt context.

### 6.2.1 XcpInit: Initialization of the XCP Protocol Layer

XcpInit

Prototype	
Single Channel	
Single Receive Channel	void <b>XcpInit</b> ( void )
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
-	-
Functional Description	
This services 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	
<ul style="list-style-type: none"> <li>■ This service function has to be called after the initialization of XCP Transport Layer.</li> <li>■ The interrupts have to be disabled while this service function is called. This function should be called during initialization of the ECU. Therefore the interrupt should have been enabled before.</li> </ul>	

### 6.2.2 XcpEvent: Handling of a data acquisition event channel

XcpEvent

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>XcpEvent</b> ( vuint8 event )
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
event	Number of event channels to process <b>The event channel numbers have to start at 0 and have to be continuous. The range is: 0..x</b>
Return code	

uint8	XCP_EVENT_NO :	Inactive (DAQ not running, Event not configured)
	XCP_EVENT_DAQ :	DAQ active */
	XCP_EVENT_DAQ_OVERRUN :	DAQ queue overflow
	XCP_EVENT_STIM :	STIM active
	XCP_EVENT_STIM_OVERRUN :	STIM data not available
<b>Functional Description</b>		
<p>Calling <code>XcpEvent</code> with a particular event channel number triggers the sampling and transmission of all DAQ lists that are assigned to this event channel.</p> <p>The event channels are defined by the ECU developer in the application program. An MCS (e.g. CANape) must know about the meaning of the event channel numbers. These are usually described in the tool configuration files or in the interface specific part of the ASAM MC2 (ASAP2) database.</p> <p><u>Example:</u></p> <p>A motor control unit may have a 10ms, a 100ms and a crank synchronous event channel. In this case, the three <code>XcpEvent</code> calls have to be placed at the appropriate locations in the ECU's program:</p> <pre>xcpEvent (0); /* 10ms cycle */ xcpEvent (1); /* 100ms cycle */ xcpEvent (2); /* Crank synchronous cycle */</pre>		
<b>Particularities and Limitations</b>		
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly and XCP is in connected state.</li> <li>■ Data acquisition has to be enabled: XCP_ENABLE_DAQ has to be defined</li> </ul>		

### 6.2.3 XcpStimEventStatus: Check data stimulation events

#### XcpStimEventStatus

<b>Prototype</b>	
Single Channel	
Single Receive Channel	uint8 <b>XcpStimEventStatus</b> ( uint8 event, uint8 action )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
event	Event channel number
action	STIM_CHECK_ODT_BUFFER : check ODT buffer STIM_RESET_ODT_BUFFER :
<b>Return code</b>	
uint8	1 : new stimulation data is available
<b>Functional Description</b>	

Check if a 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

## 6.2.4 XcpBackground: Background calculation of checksum

### XcpBackground

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>XcpBackground</b> ( 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	
<p>If the XCP command for the calculation of the memory checksum has to be used for large memory areas, it might not be appropriate to block the processor for a long period of time. Therefore, the checksum calculation is divided into smaller sections that are handled in <code>XcpBackground</code>.</p> <p>Therefore <code>XcpBackground</code> should be called periodically whenever the ECU's CPU is idle.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly</li> </ul>	

## 6.2.5 XcpSendEvent: Transmission of event codes

### XcpSendEvent

Prototype	
Single Channel	
Single Receive Channel	void <b>XcpSendEvent</b> ( 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	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly and XCP is in connected state.</li> <li>■ Data acquisition has to be enabled: XCP_ENABLE_SEND_EVENT has to be defined</li> </ul>	

## 6.2.6 XcpPutchar: Put a char into a service request packet

XcpPutchar

Prototype	
Single Channel	
Single Receive Channel	void <b>XcpPutchar</b> (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	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly and XCP is in connected state.</li> <li>■ The switch XCP_ENABLE_SERV_TEXT_PUTCHAR has to be defined</li> </ul>	

## 6.2.7 XcpPrint: Transmission of a service request packet

XcpPrint

Prototype
-----------



Single Channel	
Single Receive Channel	void <b>XcpPrint</b> ( MEMORY_ROM vuint8 *str )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
str	pointer to a string that is terminated by 0x00
<b>Return code</b>	
-	-
<b>Functional Description</b>	
Transmission of a service request packet (SERV). The string <code>str</code> is sent via service request packets. The string has to be terminated by 0x00.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly and XCP is in connected state.</li> <li>■ The switch <code>XCP_ENABLE_SERV_TEXT_PRINT</code> has to be defined</li> </ul>	

## 6.2.8 XcpDisconnect: Disconnect from XCP master

### XcpDisconnect

<b>Prototype</b>	
Single Channel	
Single Receive Channel	void <b>XcpDisconnect</b> ( void )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
str	pointer to a string that is terminated by 0x00
<b>Return code</b>	
-	-
<b>Functional Description</b>	
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.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly and XCP is in connected state.</li> </ul>	

### 6.2.9 XcpSendCrm: Transmit response or error packet

XcpSendCrm

Prototype	
Single Channel	
Single Receive Channel	void <b>XcpSendCrm</b> ( 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	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly, XCP is in connected state an a command packet (CMD) has been received..</li> </ul>	

### 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

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 <b>XcpCommand</b> ( 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 packet this function has to be called. The parameter is a pointer to the XCP protocol message, which must be extracted from the XCP protocol packet.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has to be initialized correctly.</li> </ul>	

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

#### XcpSendCallBack

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>XcpSendCallBack</b> ( 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	
<p>The XCP Protocol Layer does not call <code>ApplXcpSend</code> again, until <code>XcpSendCallBack</code> has confirmed the successful transmission of the previous message. <code>XcpSendCallBack</code> transmits pending data acquisition messages by calling <code>ApplXcpSend</code> again.</p> <p>Note that if <code>XcpSendCallBack</code> is called from inside <code>ApplXcpSend</code> a recursion occurs, which assumes enough space on the call stack.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly.</li> </ul>	

## 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 <b>ApplXcpSend</b> ( vuint8 len, MEMORY_ROM BYTEPTR msg )
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
len	Length of message data
msg	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	
■ ApplXcpSend is not defined as macro	

### 6.4.2 ApplXcpInit: Perform XCP Transport Layer initialization

ApplXcpInit

Prototype	
Single Channel	
Single Receive Channel	void <b>ApplXcpInit</b> ( void )
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	

-	-
<b>Return code</b>	
-	-
<b>Functional Description</b>	
Initializations of the XCP Transport Layer. In the XCP on CAN implementation by Vector is this function required if no transmit queue is used.	
<b>Particularities and Limitations</b>	
■ <code>ApplXcpInit</code> is not defined as macro	

### 6.4.3 ApplXcpBackground: XCP Transport Layer background operations

#### ApplXcpBackground

<b>Prototype</b>	
Single Channel	
Single Receive Channel	void <b>ApplXcpBackground</b> ( void )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
-	-
<b>Return code</b>	
-	-
<b>Functional Description</b>	
Performs background operations of the XCP Transport Layer. In the XCP on CAN implementation by Vector is this function required if no transmit queue is used.	
<b>Particularities and Limitations</b>	
■ <code>ApplXcpBackground</code> is not defined as macro	

### 6.4.4 ApplXcpInterruptEnable: Enable interrupts

#### ApplXcpInterruptEnable

<b>Prototype</b>	
Single Channel	
Single Receive Channel	void <b>ApplXcpInterruptEnable</b> ( void )
Multi Channel	
Indexed	not supported

Code replicated	not supported
<b>Parameter</b>	
-	-
<b>Return code</b>	
-	-
<b>Functional Description</b>	
Enabling of the global interrupts.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly</li> <li>■ The function <code>ApplXcpInterruptEnable</code> can be overwritten by the macro <code>ApplXcpInterruptEnable</code>.</li> </ul>	

#### 6.4.5 ApplXcpInterruptDisable: Disable interrupts

##### ApplXcpInterruptDisable

<b>Prototype</b>	
Single Channel	
Single Receive Channel	void <b>ApplXcpInterruptDisable</b> ( void )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
-	-
<b>Return code</b>	
-	-
<b>Functional Description</b>	
Disabling of the global interrupts.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly</li> <li>■ The function <code>ApplXcpInterruptDisable</code> can be overwritten by the macro <code>ApplXcpInterruptDisable</code>.</li> </ul>	

## 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.

### 6.5.1 ApplXcpGetPointer: Pointer conversion

#### ApplXcpGetPointer

Prototype	
Single Channel	
Single Receive Channel	MTABYTEPTR <b>ApplXcpGetPointer</b> ( 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	
<p>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.</p> <p>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.</p> <p>This function is used for memory transfers like DOWNLOAD and UPLOAD.</p> <p><u>Example:</u></p> <p>The following code shows an example of a typical implementation of ApplXcpGetPointer:</p> <pre>MTABYTEPTR ApplXcpGetPointer( vuint8 addr_ext, vuint32 addr ) {     return (MTABYTEPTR)addr; }</pre>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ This function can be overwritten by defining <code>ApplXcpGetPointer</code> as macro.</li> </ul>	



## 6.5.2 ApplXcpGetIdData: Get MAP filenames

### ApplXcpGetIdData

Prototype	
Single Channel	
Single Receive Channel	vuint32 <b>ApplXcpGetIdData</b> ( 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	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch <code>XCP_ENABLE_VECTOR_MAPNAMES</code> has to be defined</li> </ul>	

## 6.5.3 ApplXcpGetSeed: Generate a seed

### ApplXcpGetSeed

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpGetSeed</b> ( 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 <u>XCP Professional and XPC Basic</u> RM_CAL_PAG : to unlock the resource calibration/paging RM_DAQ : to unlock the resource data acquisition <u>XCP Professional only</u> RM_STIM : to unlock the resource stimulation RM_PGM : to unlock the resource programming

seed	Pointer to RAM where the seed has to be generated to.
<b>Return code</b>	
vuint8	The length of the generated seed that is returned by <i>seed</i> .
<b>Functional Description</b>	
Generate a seed for the appropriate resource. The seed has a maximum length of MAX_CTO-2 bytes.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch <code>XCP_ENABLE_SEED_KEY</code> has to be defined</li> </ul>	

#### 6.5.4 ApplXcpUnlock: Valid key and unlock resource

ApplXcpUnlock

<b>Prototype</b>	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpUnlock</b> (MEMORY_ROM vuint8 *key, MEMORY_ROM vuint8 length )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
key	Pointer to the key.
length	Length of the key.
<b>Return code</b>	
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
<b>Functional Description</b>	
Check the key and return the resource that has to be unlocked. Only one resource may be unlocked at one time.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch <code>XCP_ENABLE_SEED_KEY</code> has to be defined</li> </ul>	

### 6.5.5 ApplXcpCheckReadEEPROM: Check read access from EEPROM

#### ApplXcpCheckReadEEPROM

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpCheckReadEEPROM</b> (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 <code>size</code> data byte are read from <code>addr</code> and written to <code>data</code> .	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch <code>XCP_ENABLE_READ_EEPROM</code> has to be defined</li> </ul>	

### 6.5.6 ApplXcpCheckWriteEEPROM: Check write access to the EEPROM

#### ApplXcpCheckWriteEEPROM

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpCheckWriteEEPROM</b> (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 <code>addr</code> is on the EEPROM this data is written to <code>addr</code> )
<b>Return code</b>	
vuint8	XCP_CMD_OK : EEPROM written XCP_CMD_DENIED : This is not EEPROM XCP_CMD_PENDING : EEPROM write in progress, call <code>XcpSendCrm</code> when done
<b>Functional Description</b>	
<p>Checks whether the address <code>addr</code> is within the EEPROM memory. If not, the function returns <code>XCP_CMD_DENIED</code>. If it lies within, EEPROM programming is performed. The function may return during programming with <code>XCP_CMD_PENDING</code> or may wait until the programming sequence has finished and then returns with <code>XCP_CMD_OK</code>.</p> <p>If the programming sequence has finished, the <code>XcpSendCrm</code> function must be called. <code>XcpSendCrm</code> is an internal function of the XCP Protocol Layer.</p>	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch <code>XCP_ENABLE_WRITE_EEPROM</code> has to be defined</li> </ul>	

### 6.5.7 ApplXcpCheckWriteAccess: Check address for valid write access

#### ApplXcpCheckWriteAccess

<b>Prototype</b>	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpCheckWriteAccess</b> ( MTABYTEPTR address, vuint8 size)
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
address	address
size	number of bytes
<b>Return code</b>	
vuint8	0 : if access is denied >= 1 : if access is granted
<b>Functional Description</b>	
<p>Check addresses for valid write access. A write access is enabled with the <code>XCP_ENABLE_WRITE_PROTECTION</code>, it should be only used, if write protection of memory areas is required</p>	
<b>Particularities and Limitations</b>	

- 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 ApplXcpUserService: User defined command

### ApplXcpUserService

Prototype	
Single Channel	
Single Receive Channel	<code>vuint8 ApplXcpUserService ( MEMORY_ROM BYTEPTR pCmd )</code>
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
<code>pCmd</code>	Pointer to XCP command packet
Return code	
<code>vuint8</code>	<code>XCP_CMD_OK</code> : positive response <code>XCP_CMD_PENDING</code> : no response <code>XCP_CMD_SYNTAX</code> : negative response
Functional Description	
Application specific user command. Please refer to 3.10 User defined command.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch <code>XCP_ENABLE_USER_COMMAND</code> has to be defined</li> </ul>	

## 6.5.9 ApplXcpSendStall: Resolve a transmit stall condition

### ApplXcpSendStall

Prototype	
Single Channel	
Single Receive Channel	<code>vuint8 ApplXcpSendStall ( void )</code>
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-

Return code	
vuint8	0 : if not succesful > 0 : successful
Functional Description	
Resolve a transmit stall condition in <code>XcpPutchar</code> or <code>XcpSendEvent</code> .	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch <code>XCP_ENABLE_SEND_EVENT</code> or <code>XCP_ENABLE_SERV_TEXT_PUTCHAR</code> and <code>XCP_ENABLE_SEND_QUEUE</code> are defined</li> <li>■ The function can be overwritten by the macro <code>ApplXcpSendStall()</code></li> </ul>	

### 6.5.10 ApplXcpSendFlush: Flush transmit buffer

**ApplXcpSendFlush**

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpSendFlush</b> ( 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 <code>ApplXcpSend</code> . This function can be overwritten by a macro.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ The function can be overwritten by the macro <code>ApplXcpSendStall()</code></li> </ul>	

### 6.5.11 ApplXcpDisableNormalOperation: Disable normal operation of the ECU

**ApplXcpDisableNormalOperation**

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpDisableNormalOperation</b> ( MTABYTEPTR a, vuint16 size )
Multi Channel	
Indexed	not supported

Code replicated	not supported
<b>Parameter</b>	
a	Address (where the flash kernel is downloaded to)
size	Size (of the flash kernel)
<b>Return code</b>	
vuint8	XCP_CMD_OK : download of flash kernel confirmed XCP_CMD_DENIED : download of flash kernel refused
<b>Functional Description</b>	
Prior to the flash kernel download has the ECU's normal operation to be stopped in order to avoid misbehavior due to data inconsistencies.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch XCP_ENABLE_BOOTLOADER_DOWNLAOD has to be defined</li> </ul>	

### 6.5.12 ApplXcpStartBootLoader: Start of boot loader

#### ApplXcpStartBootLoader

<b>Prototype</b>	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpStartBootloader</b> ( void )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
-	-
<b>Return code</b>	
vuint8	This function should not return. 0 : negative response > 0 : positive response
<b>Functional Description</b>	
Start of the boot loader.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switch XCP_ENABLE_BOOTLOADER_DOWNLAOD has to be defined</li> </ul>	

### 6.5.13 ApplXcpReset: Perform ECU reset

#### ApplXcpReset

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

### 6.5.14 ApplXcpFlashClear: Clear flash memory

#### ApplXcpFlashClear

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpFlashClear</b> ( 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_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

### 6.5.15 ApplXcpFlashProgram: Program flash memory

#### ApplXcpFlashProgram

#### Prototype

Single Channel

Single Receive Channel	<code>vuint8 ApplXcpFlashProgram ( MEMORY_ROM BYTEPTR data, MTABYTEPTR address, vuint8 size )</code>
------------------------	--

Multi Channel

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

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

#### Parameter

data	Pointer to data
------	-----------------

address	Address
---------	---------

size	Size
------	------

#### Return code

<code>vuint8</code>	<code>XCP_CMD_OK</code> : Flash memory programming finished <code>XCP_CMD_PENDING</code> : Flash memory programming in progress. <code>XcpSendCrm</code> 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

### 6.5.16 ApplXcpDaqResume : Resume automatic data transfer

#### ApplXcpDaqResume

#### Prototype

Single Channel

Single Receive Channel	<code>vuint8 ApplXcpDaqResume ( tXcpData * xcpData )</code>
------------------------	---

Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
xcpData	Pointer to internal data of the XCP Protocol Layer
<b>Return code</b>	
vuin8	0 : failed >0 : Ok
<b>Functional Description</b>	
Resume the automatic data transfer. All communication parameters that had been stored have to be restored.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switches <code>XCP_ENABLE_DAQ</code> and <code>XCP_ENABLE_DAQ_RESUME</code> are defined</li> </ul>	

### 6.5.17 ApplXcpDaqResumeStore : Store DAQ lists for resume mode

ApplXcpDaqResumeStore

<b>Prototype</b>	
Single Channel	
Single Receive Channel	vuin8 <b>ApplXcpDaqResumeStore</b> ( MEMORY_ROM tXcpData * xcpData)
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
xcpData	Pointer to internal data of the XCP Protocol Layer
<b>Return code</b>	
vuin8	0 : failed >0 : Ok
<b>Functional Description</b>	
Store DAQ lists for DAQ resume mode.	
<b>Particularities and Limitations</b>	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switches <code>XCP_ENABLE_DAQ</code> and <code>XCP_ENABLE_DAQ_RESUME</code> are defined</li> </ul>	

### 6.5.18 ApplXcpDaqResumeClear : Clear stored DAQ lists

#### ApplXcpDaqResumeClear

Prototype	
Single Channel	
Single Receive Channel	vuint8 <b>ApplXcpDaqResumeClear</b> ( void )
Multi Channel	
Indexed	not supported
Code replicated	not supported
Parameter	
-	-
Return code	
vuin8	0 : failed >0 : Ok
Functional Description	
Clear DAQ lists that are stored during resume mode.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switches <code>XCP_ENABLE_DAO</code> and <code>XCP_ENABLE_DAO_RESUME</code> are defined</li> </ul>	

### 6.5.19 ApplXcpGetTimestamp: Returns the current timestamp

#### ApplXcpGetTimestamp

Prototype	
Single Channel	
Single Receive Channel	XcpDaqTimestampType <b>ApplXcpGetTimestamp</b> ( 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

### 6.5.20 ApplXcpGetCalPage: Get calibration page

#### ApplXcpGetCalPage

Prototype	
Single Channel	
Single Receive Channel	<code>vuint8 ApplXcpGetCalPage ( vuint8 segment, vuint8 mode )</code>
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: <code>CAL_ECU</code> : ECU access <code>CAL_XCP</code> : XCP access
Return code	
<code>vuint8</code>	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	
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switches <code>XCP_ENABLE_DAQ</code> and <code>XCP_ENABLE_DAQ_TIMESTAMP</code> are defined</li> </ul>	

### 6.5.21 ApplXcpSetCalPage: Set calibration page

#### ApplXcpSetCalPage

Prototype	
Single Channel	
Single Receive Channel	<code>vuint8 ApplXcpSetCalPage ( vuint8 segment, vuint8 page, vuint8 mode )</code>
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		
<div><div></div><div>XCP is initialized correctly and in connected state</div></div> <div><div></div><div>The switches XCP_ENABLE_DAQ and XCP_ENABLE_DAQ_TIMESTAMP are defined</div></div>		

### 6.5.22 ApplXcpCopyCalPage: Copying of calibration data pages

#### ApplXcpCopyCalPage

<b>Prototype</b>	
Single Channel	
Single Receive Channel	vuInt8 <b>ApplXcpCopyCalPage</b> ( vuInt8 srcSeg, vuInt srcPage vuInt8 destSeg, vuInt8 destPage )
Multi Channel	
Indexed	not supported
Code replicated	not supported
<b>Parameter</b>	
srcSeg	Source segment
srcPage	Source page
destSeg	Destination segment
destPage	Destination page
<b>Return code</b>	

vuint8	0 : CRC_PAGE_NOT_VALID : CRC_SEGMENT_NOT_VALID : CRC_SEGMENT_NOT_VALID :	Ok Page not available Segment not available Destination page is write protected.
<b>Functional Description</b>		
Copying of calibration data pages. The pages are copied from source to destination.		
<b>Particularities and Limitations</b>		
<ul style="list-style-type: none"> <li>■ XCP is initialized correctly and in connected state</li> <li>■ The switches XCP_ENABLE_PAGE_COPY is defined</li> </ul>		

## 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.

### 6.6.1 XcpMemCpy: Copying of a memory range

XcpMemCpy

Prototype	
Single Channel	
Single Receive Channel	void <b>XcpMemCpy</b> ( DAQBYTEPTR dest, MEMORY_ROM DAQBYTEPTR src, uint16 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	
<p>General memcpy function that copies a memory range from source to destination.</p> <p>This function is used in the inner loop of <code>XcpEvent</code> for data acquisition sampling.</p> <p>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.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly.</li> <li>■ This function can be overwritten <code>XcpMemCpy</code> is defined.</li> </ul>	

### 6.6.2 XcpMemSet: Initialization of a memory range

XcpMemSet

Prototype	
Single Channel	
Single Receive Channel	void <b>XcpMemSet</b> ( BYTEPTR p, uint16 n, uint8 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	
<p>Initialization of <i>n</i> bytes starting from address <i>p</i> with <i>b</i>.</p> <p>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.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly.</li> <li>■ This function can be overwritten <code>XcpMemSet</code> is defined.</li> </ul>	

### 6.6.3 XcpMemClear: Clear a memory range

#### XcpMemClear

Prototype	
Single Channel	
Single Receive Channel	static void <b>XcpMemClr</b> ( 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	
<p>Initialize <i>n</i> data bytes starting from address <i>p</i> with <code>0x00</code>.</p> <p>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.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly.</li> <li>■ This function can be overwritten <code>XcpMemClr</code> is defined.</li> </ul>	



### 6.6.4 XcpSendDto: Transmission of a data transfer object

XcpSendDto

Prototype	
Single Channel	
Single Receive Channel	void <b>XcpSendDto</b> ( 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	
<ul style="list-style-type: none"> <li>■ The XCP Protocol Layer has been initialized correctly and XCP is in connected state.</li> <li>■ The switch <code>XCP_ENABLE_DAQ</code> is defined</li> <li>■ This function can be overwritten by defining <code>XcpSendDto</code>.</li> </ul>	

## 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 Protocol Layer can comfortably be configured with GENy. In this case no manual configuration has to be applied to the configuration files. Therefore the following description is mainly applicable for the configuration of XCP Basic.

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.

### 7.1 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
<code>XCP_XXX_DAQ</code>	ENABLE, <b>DISABLE</b>	Enables/disables synchronous data acquisition.
<code>XCP_XXX_DAQ_PRESCALER</code>	ENABLE, <b>DISABLE</b>	Enables/disables the DAQ prescaler.
<code>XCP_XXX_DAQ_OVERRUN_INDICATION</code>	ENABLE, <b>DISABLE</b>	Enables/disables the DAQ overrun detection.
<code>XCP_XXX_DAQ_HDR_ODT_DAQ<sup>1</sup></code>	ENABLE, <b>DISABLE</b>	Use the 2 Byte DAQ/ODT XCP Packet identification instead of PID.
<code>XCP_XXX_DAQ_PROCESSOR_INFO</code>	ENABLE, <b>DISABLE</b>	Plug & play mechanism for the data acquisition processor.
<code>XCP_XXX_DAQ_RESOLUTION_INFO</code>	ENABLE, <b>DISABLE</b>	Plug & play mechanism for the data acquisition resolution.
<code>XCP_XXX_DAQ_EVENT_INFO</code>	ENABLE, <b>DISABLE</b>	Plug & play mechanism for the event definitions
<code>XCP_XXX_DAQ_TIMESTAMP</code>	ENABLE, <b>DISABLE</b>	DAQ timestamps
<code>XCP_XXX_DAQ_TIMESTAMP_FIXED</code>	ENABLE, <b>DISABLE</b>	Send always DTO Packets in time stamped mode.
<code>XCP_XXX_SEED_KEY</code>	ENABLE, <b>DISABLE</b>	Seed & key access protection.
<code>XCP_XXX_CHECKSUM</code>	ENABLE, <b>DISABLE</b>	Calculation of checksum.
<code>XCP_XXX_PARAMETER_CHECK</code>	ENABLE, <b>DISABLE</b>	Parameter check.

<sup>1</sup> 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_SEND_QUEUE	ENABLE, <b>DISABLE</b>	Transmission send queue. (should be used in conjunction with synchronous data acquisition and stimulation)
XCP_XXX_SEND_EVENT	ENABLE, <b>DISABLE</b>	Transmission of event packets (EV)
XCP_XXX_USER_COMMAND	ENABLE, <b>DISABLE</b>	User defined command
XCP_XXX_COMM_MODE_INFO	ENABLE, <b>DISABLE</b>	Communication mode info
XCP_XXX_CALIBRATION_PAGE	ENABLE, <b>DISABLE</b>	Calibration data page switching
XCP_XXX_PAGE_INFO	ENABLE, <b>DISABLE</b>	Calibration data page plug & play mechanism
XCP_XXX_PAGE_FREEZE	ENABLE, <b>DISABLE</b>	Calibration data page freezing
XCP_XXX_PAGE_COPY	ENABLE, <b>DISABLE</b>	Calibration data page copying
XCP_XXX_DPRAM	ENABLE, <b>DISABLE</b>	Supports the usage of dual port RAM

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, <b>DISABLE</b>	Enables/disables the slave block transfer.
XCP_XXX_BLOCK_DOWNLOAD	ENABLE, <b>DISABLE</b>	Enables/disables the master block transfer.
XCP_XXX_WRITE_PROTECTION	ENABLE, <b>DISABLE</b>	Write access to RAM
XCP_XXX_READ_EEPROM	ENABLE, <b>DISABLE</b>	Read access to EEPROM
XCP_XXX_WRITE_EEPROM	ENABLE, <b>DISABLE</b>	Write access to EEPROM
XCP_XXX_PROGRAM	ENABLE, <b>DISABLE</b>	Flash programming
XCP_XXX_PROGRAM_INFO	ENABLE, <b>DISABLE</b>	Flash programming plug & play mechanism
XCP_XXX_BOOTLOADER_DOWNLOAD	ENABLE, <b>DISABLE</b>	Flash programming with a flash kernel
XCP_XXX_STIM	ENABLE, <b>DISABLE</b>	Enables/disables data stimulation. (also XCP_ENABLE_DAQ has to be defined in order to use data stimulation)
XCP_XXX_DAQ_RESUME	ENABLE, <b>DISABLE</b>	Data acquisition resume mode.
XCP_XXX_SERV_TEXT	ENABLE, <b>DISABLE</b>	Transmission of service request codes
XCP_XXX_SERV_TEXT_PUTCHAR	ENABLE, <b>DISABLE</b>	Putchar function for the transmission of service request messages
XCP_XXX_SERV_TEXT_PRINTF	ENABLE, <b>DISABLE</b>	Print function for the transmission of service request messages

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	ENABLE, <b>DISABLE</b>	Test mode that allows the output of debugging information.  Not included in XCP Professional due to multiple MISRA rule violations!

## 7.2 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	<b>8</b>	Maximum length of an XCP command transfer object (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 <sup>2</sup>	<b>8</b>	Maximum length of an XCP data transfer object (DTO). The length of the DTO can be variable. However it has to be configured according to the used XCP Transport Layer.
kXcpDaqMemSize	0 .. 0xFFFF	<b>256</b>	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 <sup>3</sup>	-	Number of available events in the slave (part of event channel plug & play mechanism) Also refer to chapter 7.6.
kXcpStimOdtCount	0 .. 0xC0	<b>0xC0</b>	Maximum number of ODTs that may be used for Synchronous Data Stimulation.
kXcpChecksumMethod	-	-	Checksum calculation method. Refer to 'Table of checksum calculation methods' for valid values.
kXcpChecksumBlockSize	1 .. 0xFFFF	<b>256</b>	Each call of <code>XcpBackground</code> 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 7.8
kXcpMaxSegment	1	<b>1</b>	Number of memory segments Also refer to chapter 7.9.
kXcpMaxPages	1 .. 2	<b>2</b>	Number of pages Also refer to chapter 7.9.

<sup>2</sup> Implementation specific range. The range is 8..0xFFFF according to XCP specification [I], [II].

<sup>3</sup> Implementation specific range. The range is 0..0xFFFE according to XCP specification [I], [II].

### 7.2.1 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
XCP_CHECKSUM_TYPE_CRC16CCITT	See CRC error detection algorithms

### 7.3 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

### 7.4 Configuration of the CPU type

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

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

## 7.5 Configuration of slave device identification

The configuration of the slave device identification and automatic session configuration is described within this chapter. Only one

### 7.5.1 Identification by ASAM-MC2 filename without path and extension

If the slave device identification is done by an 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 uint8 MEMORY_ROM kXcpStationId[] = "station ID"
```

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

### 7.5.2 Automatic session configuration with MAP filenames

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

## 7.6 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 to be defined as described below:

Constant	Range	Description
<code>kXcpMaxEvent</code>	<code>0..0xFF</code> <sup>4</sup>	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.
<code>kXcpEventName[]</code>	<code>kXcpMaxEvent</code>	List with pointers to the event channel names that are defined as strings.
<code>kXcpEventNameLength[]</code>	<code>kXcpMaxEvent</code>	Length of the event channel names without the terminating char.
<code>kXcpEventCycle[]</code>	<code>kXcpMaxEvent</code>	Cycle time of the event channels in milliseconds.
<code>kXcpEventDirection[]</code>	<code>kXcpMaxEvent</code>	Direction of the event channels. For XCP Basic valid values are: - <code>kXcpEventDirectionDaq</code> For XCP Professional valid values are: - <code>kXcpEventDirectionDaq</code> - <code>kXcpEventDirectionStim</code> - <code>kXcpEventDirectionDaqStim</code>

<sup>4</sup> Implementation specific range. The range is `0..0xFFFF` 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
};
```



## 7.7 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. Additionally

Constant	Range	Description
<code>XcpDaqTimestampType</code>	<code>vuint16</code>	Type of the timestamp: <code>vuint16</code> → size is 2 bytes
<code>kXcpDaqTimestampUnit</code>	<code>DAQ_TIMESTAMP_UNIT_1NS</code> <code>DAQ_TIMESTAMP_UNIT_10NS</code> <code>DAQ_TIMESTAMP_UNIT_100NS</code> <code>DAQ_TIMESTAMP_UNIT_1US</code> <code>DAQ_TIMESTAMP_UNIT_10US</code> <code>DAQ_TIMESTAMP_UNIT_100US</code> <code>DAQ_TIMESTAMP_UNIT_1MS</code> <code>DAQ_TIMESTAMP_UNIT_10MS</code> <code>DAQ_TIMESTAMP_UNIT_100MS</code> <code>DAQ_TIMESTAMP_UNIT_1S</code>	Unit of the timestamp (1 ns, 10 ns .. 1 s)
<code>kXcpDaqTimestampTicksPerUnit</code>	<code>0..0xFFFF</code>	Time stamp ticks per unit

## 7.8 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
<code>kXcpMaxSector</code>	<code>0..0xFF</code>	Number of available flash sectors in the slave
<code>kXcpProgramSectorStart[]</code>	<code>kXcpMaxSector</code>	List with the start addresses of the sectors
<code>kXcpProgramSectorEnd[]</code>	<code>kXcpMaxSector</code>	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,
};
```

## 7.9 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 to be defined as described below:

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

## 8 Resource Requirements

The resource requirements of the XCP Protocol Layer mainly depends 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
- Atomic bit manipulation is not supported
- Only dynamic DAQ list allocation supported
- Short Download and Program Verify not supported
- The interleaved communication model is 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
- STORE\_DAQ, CLEAR\_DAQ does not send an event message
- Entering resume mode does not send an event message
- Overload indication by an event is not supported
- The timestamp size is always WORD
- The following checksum types are not supported
  - o XCP\_CRC\_16
  - o XCP\_CRC\_32
  - o XCP\_USER\_DEFINED
- Maximum checksum block size is 0xFFFF
- STORE\_CAL is not supported -> CRC\_ERROR\_OUT\_OF\_RANGE
- The seed size and key size must be equal or less MAX\_CTO-2
- Only one segment and two pages are supported

Planned:

- Atomic bit manipulation
- 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', distributed freely via the internet, that 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
- Atomic bit manipulation
- FLASH and EEPROM Programming
- The block transfer communication mode
- Resume mode
- The transmission of service request packets
- Memory write protection

## 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 to transmit 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 5.5.

## 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 1.0 of 2005-01-17

## 12 Contact

Visit our website for more information on

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

**[www.vector-informatik.com](http://www.vector-informatik.com)**