XCP on CAN

Technical Reference

XCP on CAN Transport Layer for MICROSAR CanIf

Version 1.10.00

Status	Released

Document Information

History

Date	Version	Remarks
2007-01-26	1.00.00	ESCAN00017890: Creation of Cp_XcpOnCanAsr based on Cp_XcpOnCan
2008-08-05	1.01.00	Adaptations to AUTOSAR R3
2009-12-17	1.02.00	Support of a2l export.
2010-01-14	1.03.00	ESCAN00040120: Support MultiChannel Removed Section 3.3
2010-03-30	1.04.00	ESCAN00041935: Add feature to disable XcpOnCanAsr in serial production ECUs ESCAN00043225: Missing limitation for Multiple Identity with several CAN channels
2011-01-04	1.05.00	ESCAN00046305: AR3-297 AR3-894: Support PduInfoType instead of the DataPtr
2011-03-22	1.06.00	ESCAN00049434: Support Monitoring Hooks for AUTOSAR 4
2012-04-10	1.06.01	ESCAN00052061: Added InitMemory function to chapter 8.2 Added Option for AMD Runtime Measurement
2013-03-25	1.06.02	ESCAN00063622: Remove PduInfoType from TechRef
2013-05-27	1.07.00	ESCAN00066189: Change API CanXcp_Transmit to CanXcp_Send ESCAN00070335: Remove Monitoring Hooks defined in ASR4 ESCAN00069044: Critical Section Description is missing
2014-08-15	1.07.01	ESCAN00077233: AR3-2679: Description BCD-coded return-value of CanXcp_GetVersionInfo() in TechRef
2016-02-10	1.08.00	ESCAN00082594: CAN-FD: Document support of mode 2
2016-09-07	1.09.00	ESCAN00091770: FEAT-1980: Add Multi Client / Multi Connection support
2016-10-28	1.10.00	Editorial changes

Reference Documents

No.	Title
[1]	ASAM_XCP_Part1-Overview_V1-1-0.pdf
[2]	ASAM_AE_MCD-1-XCP_AS_CAN-Transport-Layer_V1-2-0.pdf
[3]	Technical Reference XCP Protocol Layer, Version 2.04.00
[4]	AUTOSAR_SWS_DevelopmentErrorTracer.pdf V3.0.0

Abbreviations

Abbreviations	Complete expression
A2L	File Extension for an ASAM 2MC Language File
AML	ASAM 2 Meta Language

API	Application Programming Interface
ASAM	Association for Standardization of Automation and Measuring Systems
CAN	Controller Area Network
CANape	Calibration and Measurement Data Acquisition for Electronic Control Systems
CMD	Command
CTO	Command Transfer Object
DAQ	Synchronous Data Acquisition
DLC	Data Length Code (Number of data bytes of a CAN message)
DLL	Data link layer
DTO	Data Transfer Object
ECU	Electronic Control Unit
ID	Identifier (of a CAN message)
Identifier	Identifies a CAN message
ISR	Interrupt Service Routine
MCS	Master Calibration System
Message	One or more signals are assigned to each message.
MRB	Multi receive buffer
MRC	Multi receive channel
OEM	Original equipment manufacturer (vehicle manufacturer)
RES	Command Response Packet
SRB	Single receive buffer
SERV	Service Request Packet
STIM	Stimulation
XCP	Universal Measurement and Calibration Protocol
VI	Vector Informatik GmbH

Naming Conventions

The names of the access functions provided by the CanXcp always start with a prefix that includes the characters ‘CanXcp’. The characters ‘CanXcp’ 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	
CanXcp…	It is mandatory to use all functions beginning with Xcp… These services are called by either the data link layer, XCP Protocol Layer or the application. They are e.g. used for the transmission of messages.

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

Contents

1 Overview

This document describes the features, API, configuration and integration of the CanXcp. The XCP Protocol Layer, which is already described within a separate document [3], is not covered by this document.

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

XCP on CAN is a hardware independent protocol that can be ported to almost any CAN controller. 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. 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.

2 Functional Description

2.1 Overview of the Functional Scope

The transmission and reception of XCP Packets is managed by the XCP Transport Layers.

These use the AUTOSAR CanIf for the transmission and reception of XCP Packets.

The message length depends on the PDU size and can be 8 Bytes on CAN or up to 64 Bytes on CAN-FD (Mode 2).

2.2 Reception and Transmission of XCP Packets

Upon reception of any XCP message the function

```
void Xcp_CanIfRxIndication ( PduIdType CanCanXcpRxPduId,
                             const PduInfoType *CanSduPtr )
```
(4.3.1)

is called by the CAN Interface and the XCP Packet is passed to the XCP Protocol Layer.

After the command has been processed by the Protocol Layer the XCP Response Packet is passed to the Transport Layer by the service

```
void CanXcp_Send (uint8 Xcp_Channel, uint8 len, MEMORY_ROM
        BYTEPTR msg )
```
(4.2.1)

The XCP message is transmitted via the CAN Interface's service

```
uint8 CanIf_Send ( PduIdType CanTxPduId,
                   const PduInfoType *PduInfoPtr )
```

The successful transmission is confirmed by the CAN Interface by a call of

```
void Xcp_CanIfTxConfirmation ( PduIdType CanTxPduId )
```
(4.3.2)

The confirmation is passed from the XCP on Can Transport Layer to the Protocol Layer by the XcpSendCallback.

Asynchronous XCP Packet transmission like e.g. SERV, EV and DAQ are also transmitted and confirmed by the above described sequence.

Please note that after Initialization the XCP is in PDU Mode `CANXCP_SET_OFFLINE`. Normally it is enabled by the CAN State Manager. If the State Manager is not present or a State Manager from another vendor is used this has to be enabled manually by using the following API with `CANXCP_SET_ONLINE`:

```
void CanXcp_SetPduMode ( NetworkHandleType XcpNwH,
        CanXcp_PduSetModeType PduMode )
```
(4.5.3)

If this is not done, the XCP will not send anything!

3 Integration into the Application

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

3.1 Files

The XCP on CAN Transport Layer consists of the following files:

Files of the XCP on CAN Transport Layer		
`CanXcp.c`	XCP on CAN Transport Layer. This file **must not** be changed by the user!	
`CanXcp.h`	API of the XCP on CAN Transport Layer. This file **must not** be changed by the application! This file has to be included prior to `XcpProf.h`.	
`CanXcp_Types.h`	Type definitions of the XCP on CAN Transport Layer. This file **must not** be changed by the application! This file is handled internally and must not be included separately.	

Additionally the following files are generated by the generation tool GENy.

Files generated by GENy		
`CanXcp_Cfg.h`	Configuration file for XCP on CAN. External declarations for the parameters.	
`CanXcp_Lcfg.c`	Link time parameter definition for the XCP on CAN.	
`CanXcp_PBcfg.c`	Post build parameter definition for the XCP on CAN.	

Note that all files of XCP on CAN must not be changed manually except if not GENy is used for he configuration of the AUTOSAR CAN Interface. In this case only the generated files `CanXcp_Cfg.h`, `CanXcp_Lcfg.c` and `CanXcp_PBcfg.c` are relevant.

3.2 Version Changes

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

3.3 MainFunction

Normally the CanXcp_MainFunction is called cyclically by the Schedule Manager (SchM). If no SchM is present or a 3rd party SchM is used the CanXcp_MainFunction must be called cyclically by the application. The purpose of the MainFunction is to send queued messages that failed to be sent immediately. Therefore the call cycle is not very critical. A call cycle of 5 or 10ms is sufficient in most cases.

3.4 Critical Sections / Exclusive Areas

The XCP makes use of interrupt locking to guarantee atomic operation of critical sections. For this purpose one exclusive area is defined

- CANXCP_EXCLUSIVE_AREA_0

The exclusive area must be mapped to interrupt lock and unlock functions which can be called nested. The exclusive areas are used in the following cases:

3.4.1 CANXCP_EXCLUSIVE_AREA_0

This area is used whenever the services Xcp_Event, Xcp_SendCallBack, Xcp_MainFunction and Xcp_Command can interrupt each other.
Please read the Technical Reference XCP Protocol Layer [3] for further information.

3.5 Activation and Deactivation

The XCP provides functionality to disable the component during runtime. This is useful if the component shall remain in series production ECUs where XCP is disabled by default and can be enabled by, e.g. a Diagnosis service.

By default XCP is enabled. The functionality of the XCP can be disabled by calling the Macro:

```
XCP_DEACTIVATE()
```
(4.6.2)

It can be enabled again by calling the respective Macro:

```
XCP_ACTIVATE()
```
(4.6.1)

4 Description of the API

The API of XCP on CAN 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 XCP on CAN. This information is stored in XCP on CAN until it is not required anymore, respectively until it is changed by other operations.

Examples for calling the services of XCP on CAN can be found in the description of the services.

4.1 Version of the Source Code

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

```
CONST(uint8, CANXCP_CONST) kXcpOnCanAsrMainVersion    =
      (uint8)(CP_XCPONCANASR_VERSION >> 8);
CONST(uint8, CANXCP_CONST) kXcpOnCanSubAsrVersion     =
      (uint8)(CP_XCPONCANASR_VERSION & 0x00ff);
CONST(uint8, CANXCP_CONST) kXcpOnCanAsrReleaseVersion =
      (uint8)(CP_XCPONCANASR_RELEASE_VERSION);
```

Example
Version 1.00.00 is registered as:

```
kXcpOnCanAsrMainVersion = 0x01;
kXcpOnCanAsrSubVersion = 0x00;
kXcpOnCanAsrReleaseVersion = 0x00;
```

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

4.2 XCP on CAN Transport Layer services called by the Protocol Layer

The following XCP on CAN Transport Layer functions are called by the Protocol Layer. The API of these functions can be found in the header of the XCP on CAN component.

4.2.1 CanXcp_Send: Transmission of XCP Packets

CanXcp_Send

Prototype
`void CanXcp_Send ( uint8 Xcp_Channel, uint8 len, const uint8* msg )`

Parameter	
`Xcp_Channel`	Logical channel of the protocol layer. Depending whether Multi Client is enabled this parameter will always be 0 (Multi Client disabled) or reflect the logical Xcp channel (Multi Client enabled).
`len`	Length of the XCP Packet that has to be transmitted. (with len = 1 ... 8 on CAN and 1..64 on CAN-FD)
`msg`	Pointer to the XCP Packet data.
Return code	
-	-
Functional Description	
Request for the transmission of a DTO or CTO message.	
Particularities and Limitations	
■ Not re-entrant ■ Call context of: XcpEvent, XcpBackground and context of CAN Interface	

4.2.2 CanXcp_MainFunction: Main Function of XCP on CAN Transport Layer

CanXcp_MainFunction

Prototype
`void` **`CanXcp_MainFunction`** `( void )`
Parameter
- | -
Return code
- | -
Functional Description
Main function of XCP on CAN Transport Layer.
Particularities and Limitations
■ Not re-entrant

4.3 XCP Transport Layer for CAN services called by the CAN Interface

The following XCP on CAN Transport Layer functions are called by the AUTOSAR CAN Interface. The API of these functions can be found in the header of the CAN Interface's parameter file.

4.3.1 Xcp_CanIfRxIndication: XCP Message Indication Function

Xcp_CanIfRxIndication

Prototype	
`void Xcp_CanIfRxIndication ( PduIdType CanCanXcpRxPduId, const PduInfoType * PduInfoPtr)`	
Parameter	
`CanCanXcpRxPduId`	Target PDU ID of CAN L-PDU that has been received
`PduInfoPtr`	Contains pointer and length to received XCP L-SDU
Return code	
-	–
Functional Description	
Rx Indication for reception of CTO and DTO Packets. This function is configured in the generation tool.	
Particularities and Limitations	
■ Not re-entrant ■ Call context of CAN Interface	

4.3.2 Xcp_CanIfTxConfirmation: XCP Message Confirmation

Xcp_CanIfTxConfirmation

Prototype	
`void Xcp_CanIfTxConfirmation ( PduIdType CanTxPduId )`	
Parameter	
`CanTxPduId`	PDU ID of CAN L-PDU transmitted successfully
Return code	
-	-
Functional Description	
Tx Confirmation for successful transmission of CTO and DTO Packets. This function is configured in the generation tool.	
Particularities and Limitations	
■ Not re-entrant ■ Call context of CAN Interface	

4.4 XCP Protocol Layer services called by the XCP on CAN Transport Layer

The following XCP Protocol Layer services are called by the XCP on CAN Transport Layer:

- `void Xcp_Command( uint8 Xcp_Channel, const uint32* pCommand )`
- `void Xcp_SendCallBack( uint8 Xcp_Channel )`
- `void Xcp_SetActiveTl( uint8 Xcp_Channel, uint8 Tl )`
- `uint8 Xcp_GetActiveTl( uiont8 Xcp_Channel )`
- `uint8 Xcp_GetSessionStatus( uint8 Xcp_Channel )`

For a description of the API and the functionality of these functions please refer to the Technical Reference XCP Protocol Layer [3].

4.5 XCP on CAN Transport Layer services called by other layers

The following XCP on CAN Transport Layer functions have to be called from other layers.

4.5.1 CanXcp_Init: Initialization of XCP on CAN Transport Layer

CanXcp_Init

Prototype	
`void` **`CanXcp_Init`** `(const CanXcp_ConfigType * ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to the post build configuration
Return code	
-	-
Functional Description	
Initialization of the XCP on Transport Layer.	
Particularities and Limitations	
■ Must be called during system initialization ■ Not re-entrant ■ Parameter '`ConfigPtr`' is only evaluated for post build configurations.	

4.5.2 CanXcp_GetVersionInfo: Request version information of XCP on CAN Transport Layer

CanXcp_Init

Prototype	
`void` **`CanXcp_GetVersionInfo`** `(Std_VersionInfoType * Versioninfo )`	
Parameter	
`Versioninfo`	Pointer to store the version information

Return code	
-	-

Functional Description

CanXcp_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.

Particularities and Limitations

- none

4.5.3 CanXcp_SetPduMode: set Transmission mode

CanXcp_SetPduMode

Prototype

```
void CanXcp_SetPduMode( NetworkHandleType XcpNwH, CanXcp_PduSetModeType PduMode );
```

Parameters [in/out/both]	
XcpNwH [in]	The Network Handle which is usually 0
PduMode	This is the new state. It can either be ■ CANXCP_SET_ONLINE ■ CANXCP_SET_OFFLINE

Return code	
–	–

Service ID	
Service ID	7

Functional Description

With this service it is possible to prevent transmission of XCP frames, e.g. when the bus is offline. The frames are not lost but stored in the Send Queue until overrun occurs or transmission is enabled again.

Preconditions

-

Postconditions

-

Particularities and Limitations

- Call context: task level
- Not re-entrant
- Synchronous

4.6 Macros

4.6.1 XCP_ACTIVATE: Enable the Protocol and Transport Layer

XCP_ACTIVATE

Prototype	
`XCP_ACTIVATE();`	
Parameters [in/out/both]	
`Command [in]`	–
Return code	
–	–
Service ID	
Service ID	-
Functional Description	
With this service it is possible to enable all functionality of the XCP Protocol and Transport Layer during runtime to prevent erroneous execution.	
Preconditions	
-	
Postconditions	
-	
Particularities and Limitations	
■ Call context: task level ■ Not re-entrant ■ Synchronous	

4.6.2 XCP_DEACTIVATE: Disable the Protocol and Transport Layer

XCP_DEACTIVATE

Prototype	
`XCP_DEACTIVATE();`	
Parameters [in/out/both]	
`Command [in]`	–
Return code	
–	–
Service ID	
Service ID	-
Functional Description	
With this service it is possible to lock all functionality of the XCP Protocol and Transport Layer during runtime to prevent erroneous execution.	
Preconditions	
-	
Postconditions	
-	

Particularities and Limitations
■ Call context: task level ■ Not re-entrant ■ Synchronous

4.6.3 A2L File

GENy exports an a2l file for easier configuration of the XCP Master (e.g. CANape). This file is called CanXCPAsr.a2l and contains the XCP_ON_CAN IF_DATA section which can be included by a master template a2l file.

Hint!

▶ The following abstract can be used to include the generated a2l:

```
/begin IF_DATA XCP
  ……..
  /include "CanXCPAsr.a2l"
/end IF_DATA
```

5 Limitations

5.1.1 Assignment of CAN identifiers to DAQ Lists is not supported

The assignment of CAN identifiers to DAQ lists is not supported. There is only one CAN identifier for CMD/STIM and one CAN identifier for RES/EV/DAQ. The command GET/SET_DAQ_ID is not supported.

5.1.2 Detection of all XCP Slaves within a Network

The detection of all XCP slaves within a network with the command GET_SLAVE_ID is not supported.

5.1.3 Multiple Identity only supported for single channel configuration

Multiple Identities for AUTOSAR XCP on CAN is only available for a single CAN channel configuration.

6 FAQ

6.1 Transmit Queue of CAN Interface is Disabled

FAQ
How to operate XCP on CAN if the transmit queue of CanIf is disabled.

If the transmit queue of CanIf is disabled, at any time it might not be possible to transmit the XCP Slave message due to an on-going message transmission. Therefore the message transmission might have to be requested several times.
This is done with the service `CanXcp_MainFunction()`. This service has to be called cyclically with a recommended call cycle of 1ms. The faster it gets called the faster the XCP Slave message will participate in the arbitration on the bus.

6.2 Additions & Extension

6.2.1 CanXcp_InitMemory

The function CanXcp_InitMemory is an extension to AUTOSAR and must be used when the memory is not initialized by the startup code.

7 Contact

Visit our website for more information on

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

www.vector.com