

XCP on CAN

Technical Reference

XCP on CAN Transport Layer for MICROSAR CanIf

Version 1.06.00

Authors:	Frank Triem, Sven Hesselmann, Andreas Herkommer
Version:	1.06.00
Status:	released (in preparation/completed/inspected/released)

1 Document Information

1.1 History

Author	Date	Version	Remarks
Frank Triem	2007-01-26	1.00.00	ESCAN00017890: Creation of Cp_XcpOnCanAsr based on Cp_XcpOnCan
Sven Hesselmann	2008-08-05	1.01.00	Adaptations to AUTOSAR R3
Mario Kunz	2009-12-17	1.02.00	Support of a2l export.
Andreas Herkommer	2010-01-14	1.03.00	ESCAN00040120: Support MultiChannel Removed Section 3.3
Andreas Herkommer	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
Andreas Herkommer	2011-01-04	1.05.00	ESCAN00046305: AR3-297 AR3-894: Support PduInfoType instead of the DataPtr
Andreas Herkommer	2011-03-22	1.06.00	ESCAN00049434: Support Monitoring Hooks for AUTOSAR 4

1.2 Reference Documents

Index	Document
[1]	XCP -Part 1 – Overview, Version 1.0 of 2003-04-08
[2]	XCP -Part 2- Protocol Layer Specification, Version 1.0 of 2003-04-08
[3]	XCP -Part 5- Example Communication Sequences, Version 1.0 of 2003-04-08
[4]	Technical Reference XCP Protocol Layer, Version 1.0 of 2005-01-17
[5]	AN-AND-1-108 Glossary of CAN Protocol Terminology http://www.vector-informatik.de
[6]	AUTOSAR_SWS_DevelopmentErrorTracer.pdf V3.0.0

1.3 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

Also refer to [5] for a list of common abbreviations and terms.

1.4 Naming Conventions

The names of the access functions provided by the XCP on CAN Transport Layer for MICROSAR CanIf 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...	<p>It is mandatory to use all functions beginning with Xcp...</p> <p>These services are called by either the data link layer, XCP Protocol Layer or the application.</p> <p>They are e.g. used for the transmission of messages.</p>



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

1	Document Information	2
1.1	History	2
1.2	Reference Documents	2
1.3	Abbreviations	2
1.4	Naming Conventions	3
2	Overview	7
3	Functional Description	8
3.1	Overview of the Functional Scope	8
3.2	Reception and Transmission of XCP Packets	8
3.3	Activation and Deactivation	9
3.4	Hook Functions (AUTOSAR 4 only)	9
4	Integration into the Application	10
4.1	Files	10
4.2	Version Changes	10
4.3	Integration of XCP on CAN into the Application	11
5	Description of the API	13
5.1	Version of the Source Code	13
5.2	XCP on CAN Transport Layer services called by the Protocol Layer	14
5.2.1	CanXcp_Transmit: Transmission of XCP Packets	14
5.2.2	CanXcp_MainFunction: Main Function of XCP on CAN Transport Layer ..	14
5.3	XCP Transport Layer for CAN services called by the CAN Interface	15
5.3.1	CanXcp_RxIndication: XCP Message Indication Function ("Use Pdu Info Type" disabled)	15
5.3.2	CanXcp_RxIndication: XCP Message Indication Function ("Use Pdu Info Type" enabled)	15
5.3.3	CanXcp_TxConfirmation: XCP Message Confirmation	16
5.4	XCP Protocol Layer services called by the XCP on CAN Transport Layer	16
5.5	XCP on CAN Transport Layer services called by other layers	16
5.5.1	CanXcp_Init: Initialization of XCP on CAN Transport Layer	16
5.5.2	CanXcp_GetVersionInfo: Request version information of XCP on CAN Transport Layer	17
5.5.3	CanXcp_Control: Functionality lock	17
5.5.4	CanXcp_SetPduMode: set Transmission mode	18

5.6	Macros	18
5.6.1	XCP_ACTIVATE: Enable the Protocol and Transport Layer	18
5.6.2	XCP_DEACTIVATE: Disable the Protocol and Transport Layer.....	19
6	Configuration of XCP on CAN.....	20
6.1	Configuration of XCP on CAN and CanIf with GENy	20
6.1.1	Main Configuration Page.....	21
6.1.2	Channel Configuration Page	22
6.1.3	A2L File	23
7	Limitations	24
7.1.1	Variable Length of XCP Packets is not supported	24
7.1.2	Assignment of CAN identifiers to DAQ Lists is not supported.....	24
7.1.3	Detection of all XCP Slaves within a Network.....	24
7.1.4	Multiple Identity only supported for single channel configuration.....	24
8	FAQ.....	25
8.1	Transmit Queue of CAN Interface is Disabled	25
9	Contact.....	26

Illustrations

Figure 4-1	Integration of XCP on CAN into the application	11
Figure 6-1	Component selection	20
Figure 6-2	Main configuration page of XCP on CAN Transport Layer	21
Figure 6-3	Channel configuration page of XCP on CAN Transport Layer.....	22

2 Overview

This document describes the features, API, configuration and integration of the XCP on CAN Transport Layer for MICROSAR CanIf. The XCP Protocol Layer, which is already described within a separate document [4], 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.

3 Functional Description

3.1 Overview of the Functional Scope

The transmission and reception of XCP Packets is managed by the XCP Transport Layers. The XCP on CAN Transport Layer for MICROSAR CanIf uses the AUTOSAR CanIf for the transmission and reception of XCP Packets. Variable message length is not supported and therefore the XCP Transport Layer ensures that all transmitted XCP messages have the same DLC of 8 Bytes.

3.2 Reception and Transmission of XCP Packets

Upon reception of any XCP message the function

```
void CanXcp_RxIndication ( PduIdType CanCanXcpRxPduId,  
                           const uint8 * CanSduPtr )
```

 (5.3.1)

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_Transmit (vuint8 len, MEMORY_ROM BYTEPTR msg )
```

 (5.2.1)

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

```
vuint8 CanIf_Transmit ( PduIdType CanTxPduId,  
                       const PduInfoType *PduInfoPtr )
```

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

```
void CanXcp_TxConfirmation ( PduIdType CanTxPduId )
```

 (5.3.3)

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 API:

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

 (5.5.4)

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

3.3 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 CanXcp can be disabled by calling the Macro:

```
XCP_DEACTIVATE() (5.6.2)
```

It can be enabled again by calling the respective Macro:

```
XCP_ACTIVATE() (5.6.1)
```

Please Note that the XCP will continue where it was stopped, so a running measurement will also continue without the Master knowing about it. Therefore it is safer to call XcpInit() and CanXcp_Init() in order to re-initialize the XCP to reactivate it.

3.4 Hook Functions (AUTOSAR 4 only)

With the AUTOSAR 4 edition it is now possible to make use of Hook functions that are called at the beginning and end of each function. These hook functions are realized as empty defines in the file CanXcp_Hooks.h and therefore do not use any resources.

They can be customized to user defined functions which must be added to this file manually.



Example

```
#define CanXcpHook_OnBegin_InitMemory MyHookFunction_Begin();  
#define CanXcpHook_OnEnd_InitMemory MyHookFunction_End();
```



Note

The Hook functions are called after the DET. This means that if you get a DET error the function might be exited and no Hook function is called.





Also the Hook functions are Pre-Compile time configurable. That means that they can not be modified for Library deliveries.

4 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.

4.1 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.	
CanXcp_Hooks.h	This file is only available in the AUTOSAR 4.0 version only and can be edited manually to make use of the Hook functions.	

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 the 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.

4.2 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.

4.3 Integration of XCP on CAN into the Application

XCP on CAN comprises the XCP Protocol Layer in conjunction with the XCP on CAN Transport Layer for MICROSAR CanIf.

Note that the CAN Interface, 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:

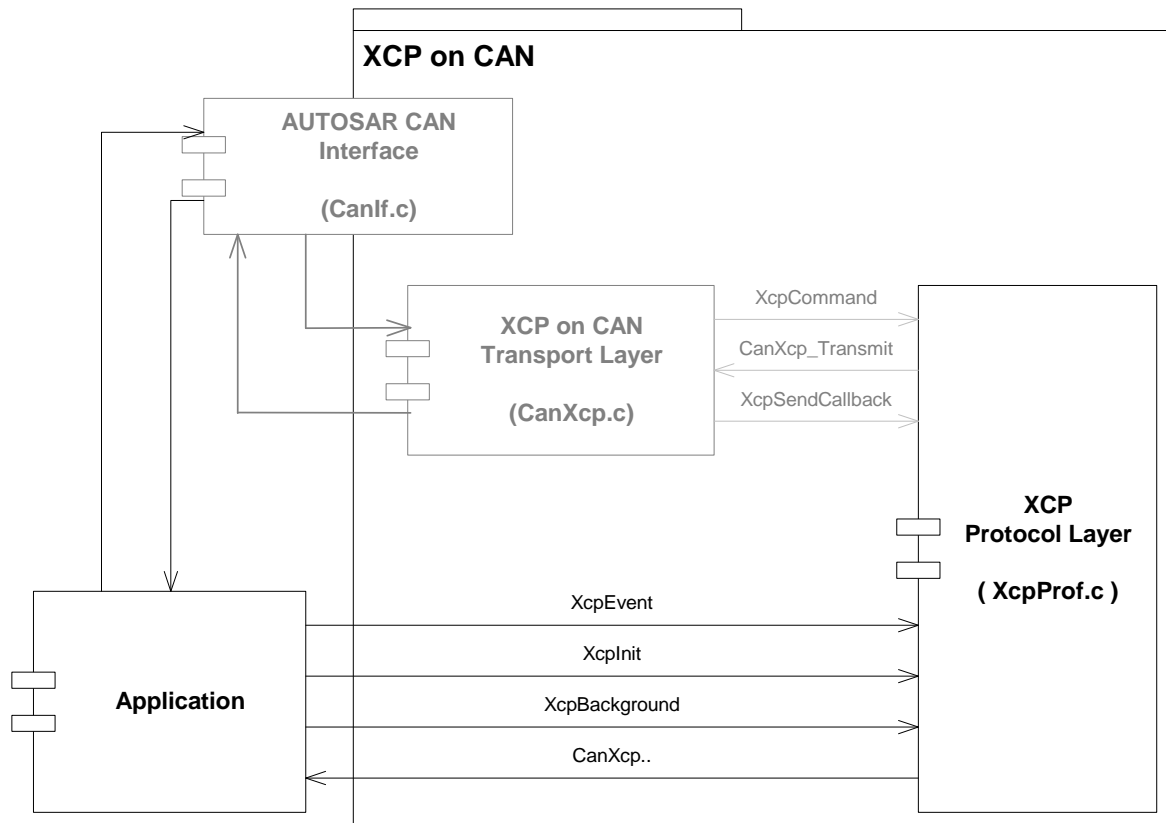


Figure 4-1 Integration of XCP on CAN into the application



Practical Procedure

The integration of XCP on CAN is done by following these steps:

1. Configure XCP on CAN in the generation tool GENy and generate.
2. Include the include header file `XcpProf.h` into all modules that access the XCP on CAN services or provide services that XCP on CAN uses.
3. Add all source files and generated source files in the make file and link it together with the data link layer and the application.
4. Initialize the data link layer after each reset during start-up before initializing XCP on CAN (interrupts have to be disabled until the complete initialization procedure is done) by calling `XcpInit`.
5. If required call the background function `XcpBackground` cyclically.
6. Integrate the desired XCP on CAN services into your application. Call especially the function `XcpEvent(channel)` cyclic with the appropriate cycle time and channel number.

	The XCP on CAN sources must not be changed for the integration into the application.
--	--------------------------------------------------------------------------------------

5 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.

5.1 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.

5.2 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.

5.2.1 CanXcp_Transmit: Transmission of XCP Packets

CanXcp_Transmit

Prototype	
void CanXcp_Transmit (uint8 len, const uint8* msg)	
Parameter	
len	Length of the XCP Packet that has to be transmitted. (with len = 1 ... 8)
msg	Pointer to the XCP Packet data.
Return code	
-	-
Functional Description	
Request for the transmission of a DTO or CTO message.	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ Not reentrant ■ Call context of: XcpEvent, XcpBackground and context of CAN Interface 	

5.2.2 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	
<ul style="list-style-type: none"> ■ Not reentrant ■ Call context of XcpBackground 	

5.3 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.

5.3.1 CanXcp_RxIndication: XCP Message Indication Function ("Use Pdu Info Type" disabled)

CanXcp_RxIndication

Prototype	
void CanXcp_RxIndication (PduIdType CanCanXcpRxPduId, const uint8 * CanSduPtr)	
Parameter	
CanCanXcpRxPduId	Target PDU ID of CAN L-PDU that has been received
CanSduPtr	Pointer to received L-SDU (payload)
Return code	
-	-
Functional Description	
Rx Indication for reception of CTO and DTO Packets. This function is configured in the generation tool.	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ Not reentrant ■ Call context of CAN Interface 	

5.3.2 CanXcp_RxIndication: XCP Message Indication Function ("Use Pdu Info Type" enabled)

CanXcp_RxIndication

Prototype	
void CanXcp_RxIndication (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	

5.3.3 CanXcp_TxConfirmation: XCP Message Confirmation

CanXcp_TxConfirmation

Prototype

```
void CanXcp_TxConfirmation ( PduIdType CanTxPduId )
```


ConfigPtr	Pointer to the post build configuration
Return code	
-	-
Functional Description	
Initialization of the XCP on Transport Layer.	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ Must be called during system initialization ■ Not reentrant ■ Parameter 'ConfigPtr' is only evaluated for post build configurations. 	

5.5.2 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	
The function writes the version information and vendor, module and instance id to the given pointer.	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ none 	

5.5.3 CanXcp_Control: Functionality lock

CanXcp_Control

Prototype	
void CanXcp_Control(uint8 command);	
Parameters [in/out/both]	
Command [in]	This is the new state. It can either be <ul style="list-style-type: none"> ■ kXcponCan_Control_Disable ■ kXcponCan_Control_Enable
Return code	
-	-
Service ID	
Service ID	6
Functional Description	
With this service it is possible to lock all functionality of the CanXCP during runtime to prevent erroneous execution.	
Preconditions	

-
Postconditions
-
Particularities and Limitations
<ul style="list-style-type: none"> ■ Call context: task level ■ Not re-entrant ■ Synchronous

5.5.4 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 <ul style="list-style-type: none"> ■ 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	
<ul style="list-style-type: none"> ■ Call context: task level ■ Not re-entrant ■ Synchronous 	

5.6 Macros

5.6.1 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. The Macros can be used instead of calling the service described in 5.5.3 Functionality lock.	
Preconditions	
-	
Postconditions	
-	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ Call context: task level ■ Not re-entrant ■ Synchronous 	

5.6.2 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. The Macros can be used instead of calling the service described in 5.5.3 Functionality lock	
Preconditions	
-	
Postconditions	
-	
Particularities and Limitations	
<ul style="list-style-type: none"> ■ Call context: task level ■ Not re-entrant ■ Synchronous 	

6 Configuration of XCP on CAN

The configuration of XCP on CAN (XCP Protocol Layer and XCP on CAN Transport Layer) is supported by the generation tool GENy.

6.1 Configuration of XCP on CAN and CanIf with GENy

If GENy is used for the configuration of both XCP on CAN and the AUTOSAR CAN Interface the whole configuration can conveniently be carried out with GENy.

No database attributes are required for the configuration of XCP on CAN.

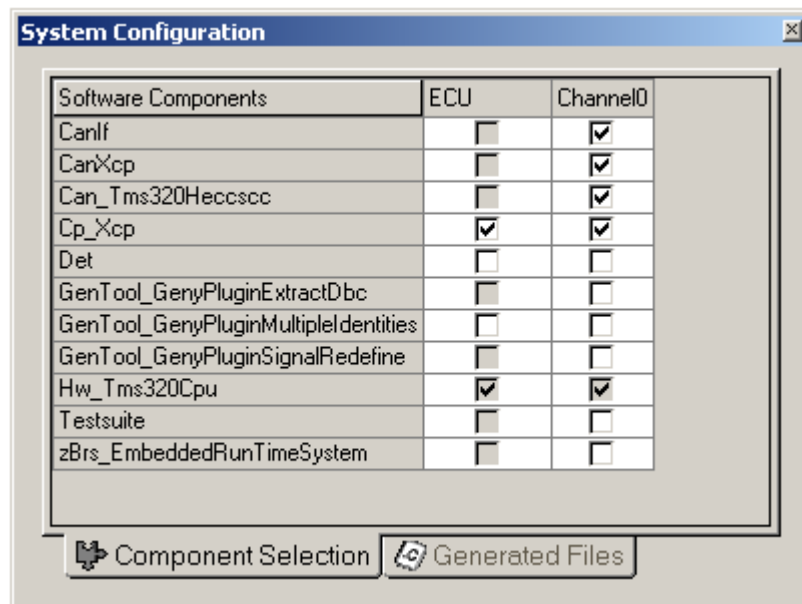


Figure 6-1 Component selection

In order to configure the XCP on CAN Transport Layer for MICROSAR CanIf (CanXcp) it has to be activated on the designated channels. The activation of the XCP Transport Layer for CAN requires the activation of the XCP Protocol Layer (Cp_Xcp).

The configuration of each component is done on separate dialogs. The XCP on CAN Transport Layer has component specific and channel specific settings that may be customized separately.

6.1.1 Main Configuration Page

Configurable Options		CanXcp
Common configuration		
Configuration Variant	Variant 1 (Pre-compile Configuration)	
Get Version Info	<input type="checkbox"/> *	
Development Error Detection	<input type="checkbox"/> *	
Production Error Detection	<input checked="" type="checkbox"/> *	
User Config File	* ...	
Module Start Address	0x0*	
Use PduInfoType	<input type="checkbox"/> *	
XCP on CAN Transport Layer		
Variable DLC	<input type="checkbox"/> *	
Multi Connection Protection	<input type="checkbox"/> *	

Figure 6-2 Main configuration page of XCP on CAN Transport Layer

Configuration options	Value	Description
XCP on CAN Transport Layer options		
Variable DLC	<input checked="" type="checkbox"/> Enable <input checked="" type="checkbox"/> Disable	Activate/Deactivate the transmission of messages with variable DLC. This option is not available yet!
Multi connection protection	<input checked="" type="checkbox"/> Enable <input checked="" type="checkbox"/> Disable	Activate/Deactivate the protection against multiple connections. Only available if XCP on CAN is used on multiple CAN channels.
Configuration Variant	<input checked="" type="checkbox"/> Variant 1 <input checked="" type="checkbox"/> Variant 2 <input checked="" type="checkbox"/> Variant 3	Select which configuration variant shall be applied to the CAN driver module. - Pre-compile Configuration: All settings can be changed and the driver has to be compiled - Link-time Configuration: All settings except precompiled ones can be changed and driver object code has to be linked - Post-build Configuration: Only post build settings can be changed and the generated data has to be flashed to the Target Please Note: If you use Multiple Identities you have to select Configuration Variant 3: Post-build!
Development Error Detection	<input checked="" type="checkbox"/> Enable <input checked="" type="checkbox"/> Disable	Detection of Development Errors like: - service call initialization - invalid channel handle - rejection of service calls All errors are reported to the Development Error Tracer (if available)
Production Error Detection	<input checked="" type="checkbox"/> Enable <input checked="" type="checkbox"/> Disable	If 'Production Error Detection' is enabled, production relevant errors are reported to the Diagnostics Event Manager (DEM). Disable this option, if no DEM is

		present in your system.
Get Version Info	<input checked="" type="checkbox"/> Enable <input checked="" type="checkbox"/> Disable	Check this box to enable function CanXcp_GetVersionInfo() to get major, minor and patch version information.
Module Start Address	<input checked="" type="checkbox"/> Address	This Value specifies the Start Address of the CanXcp Configuration. The Start Address is needed in the post-build configuration process to locate and replace the previous configuration correctly.
Use PduInfoType	<input checked="" type="checkbox"/> Enable <input checked="" type="checkbox"/> Disable	If this option is enabled the CanXcp_RxIndication function is according to AUTOSAR 3.1.5.

Table 6-1 Main configuration page of XCP on CAN Transport Layer

6.1.2 Channel Configuration Page

Configurable Options		ChannelID
<input type="checkbox"/> General Settings		
Bus System Type	CAN	
Manufacturer	Vector	
<input type="checkbox"/> Internal Properties		
Datasource	DatabaseFile	
<input type="checkbox"/> XCP on CAN Transport Layer		
Slave ID	XcpSlave	
Master ID	XcpMaster	

Figure 6-3 Channel configuration page of XCP on CAN Transport Layer

Configuration options	Value	Description
Type of bus system	Ready Only	Bus system type for the specific channel.
Manufacturer	Ready Only	Value of the database attribute 'manufacturer'.
XCP on CAN Transport Layer		
Slave Id	Tx-ID	<p>XCP Slave Identifier</p> <p>This is the ID for Response Packets and DAQ packets. Only IDs that are no IL, TP or Diag messages and that have a DLC of 8 can be selected.</p> <p>The configuration within the CanIf is automatically done by selecting the message.</p>
Master Id	RX-ID	<p>XCP Master Identifier</p> <p>This is the ID for Command Packets and STIM Packets. Only IDs that are no IL, TP or Diag messages and that have a DLC of 8 can be selected.</p> <p>The configuration within the CanIf is automatically done by selecting the message.</p>

Table 6-2 Channel configuration page of XCP on CAN Transport Layer

6.1.3 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
```

7 Limitations

7.1.1 Variable Length of XCP Packets is not supported

The XCP Protocol allows a variable length of XCP Packets. However many OEMs require that all CAN messages sent within their automotive networks have to have a static DLC. Therefore the DLC of XCP on CAN messages is always 8 and the Control Field of the XCP Tails consists of fill bytes.

7.1.2 Assignment of CAN identifiers to DAQ Lists is not supported

The assignment of CAN identifiers to DAQ lists is not supported.

7.1.3 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.

7.1.4 Multiple Identity only supported for single channel configuration

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

8 FAQ

8.1 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 ongoing message transmission. Therefore the message transmission might have to be requested several times.

This is done with the service `CanXcp_MainFunction()` that gets called by `XcpBackground()`. This service has to be called cyclic 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.

9 Contact

Visit our website for more information on

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

www.vector-informatik.com