

# **XCP on CAN**

## **Technical Reference**

XCP on CAN Transport Layer

Version 1.08

<b>Authors:</b>	Frank Triem, Sven Hesselmann
<b>Version:</b>	1.08
<b>Status:</b>	released (in preparation/completed/inspected/released)

## 1 Document Information

### 1.1 History

Author	Date	Version	Remarks
Frank Triem Klaus Emmert	2005-01-03	1.0	ESCAN00010737: Initial draft Warning Text added
Frank Triem	2005-02-28	1.1	ESCAN00011300: Manual configuration
Frank Triem	2005-06-22	1.2	ESCAN00011772: Support multiple CAN channels ESCAN00012311: Support CAN-Driver without transmit queue
Frank Triem	2005-12-19	1.3	Rework due to inspection
Frank Triem	2006-04-24	1.4	ESCAN00015915: Correct filenames
Frank Triem	2006-05-30	1.5	ESCAN00016517: Update of table of contents
Frank Triem	2006-10-26	1.6	ESCAN00017220: Documentation of reentrant capability of all functions
Sven Hesselmann	2007-09-14	1.7	Multiple Identity added
Sven Hesselmann	2008-03-19	1.07.01	Invalid reference corrected
Andreas Herkommer	2009-01-14	1.08	ESCAN00031509: Description of how to define Messages as Application Messages

### 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]	Technical Reference CAN Driver, Version 2.21 of 2003-07-29
[6]	AN-AND-1-108 Glossary of CAN Protocol Terminology <a href="http://www.vector-informatik.de">http://www.vector-informatik.de</a>

### 1.3 Abbreviations

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>CAN</b>	<b>Controller Area Network</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>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>MRB</b>	<b>Multi receive buffer</b>
<b>MRC</b>	<b>Multi receive channel</b>
<b>OEM</b>	<b>Original equipment manufacturer</b> (vehicle manufacturer)
<b>RES</b>	Command <b>Response Packet</b>
<b>SRB</b>	<b>Single receive buffer</b>
<b>SERV</b>	<b>Service Request Packet</b>
<b>STIM</b>	<b>Stimulation</b>
<b>XCP</b>	Universal Measurement and <b>Calibration Protocol</b>
<b>VI</b>	<b>Vector Informatik GmbH</b>

Also refer to [6] for a list of common abbreviations and terms.

## 1.4 Naming conventions

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

Naming conventions	
Xcp...	It is mandatory to use all functions beginning with Xcp... These services are called by either the data link layer, XCP Protocol Layer or the application. They are e.g. used for the transmission of messages.
ApplXcp	The functions, starting with ApplXcp... are functions that are provided by the application and are called by the XCP Transport Layer for CAN. These services are user callback functions that are application specific and have to be implemented depending on the application.



### 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>Document Information .....</b>	<b>2</b>
1.1	History .....	2
1.2	Reference Documents .....	2
1.3	Abbreviations .....	3
1.4	Naming conventions.....	4
<b>2</b>	<b>Overview .....</b>	<b>7</b>
<b>3</b>	<b>Functional Description .....</b>	<b>8</b>
3.1	Overview of the functional scope .....	8
3.2	Reception and transmission of XCP packets .....	8
3.3	Support of multiple CAN channels .....	8
<b>4</b>	<b>Integration into the application.....</b>	<b>9</b>
4.1	Files.....	9
4.2	Version changes .....	9
4.3	Integration of XCP on CAN into the application .....	10
<b>5</b>	<b>Description of the API.....</b>	<b>12</b>
5.1	Version of the source code .....	12
5.2	XCP Transport Layer for CAN services called by the Protocol Layer .....	13
5.2.1	ApplXcpSend: Transmission of XCP Packets.....	13
5.2.2	ApplXcpInit: Initialization of XCP Transport Layer for CAN.....	13
5.2.3	ApplXcpBackground: Background task of XCP Transport Layer for CAN.....	14
5.3	XCP Transport Layer for CAN services called by the CAN-Driver .....	15
5.3.1	XcpPreCopy: XCP message precopy function.....	15
5.3.2	XcpConfirmation: XCP message confirmation .....	16
5.4	XCP Protocol Layer services called by the Transport Layer for CAN .....	16
5.5	CAN-Driver services called by the Transport Layer for CAN .....	16
<b>6</b>	<b>Configuration of XCP on CAN.....</b>	<b>17</b>

- 6.2.2 XCP on CAN uses multiple CAN channels ..... 24
- 7 Limitations ..... 25**
  - 7.1.1 Variable length of XCP Packets is not supported ..... 25
  - 7.1.2 Assignment of CAN identifiers to DAQ lists is not supported ..... 25
  - 7.1.3 Detection of all XCP slaves within a network ..... 25
  - 7.1.4 Channel API ..... 25
  - 7.1.5 Multiple Identity only supported for single channel configuration ..... 25
- 8 FAQ ..... 26**
  - 8.1 Transmit queue of CAN-Driver is disabled ..... 26
- 9 Contact ..... 27**

**Illustrations**

- figure 4-1 Integration of XCP on CAN into the application ..... 10
- figure 6-1 Component selection ..... 17
- figure 6-2 Main configuration page of XCP on CAN Transport Layer ..... 18
- figure 6-3 Channel configurw 12o.T0 1 Tf0T1 Tm6()-4(P on )-5(C3Transport Layer)-170(.....9.....)-2

## 2 Overview

This document describes the features, API, configuration and integration of the XCP Transport Layer for CAN. 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, 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.

## 3 Functional Description

### 3.1 Overview of the functional scope

The XCP Transport Layers manage the transmission and reception of XCP Packets.

The XCP Transport Layer for CAN makes use of the CAN-Driver to transmit and receive XCP messages. Since variable message length is not supported the XCP Transport Layer has to ensure that all sent XCP messages have the same DLC. I.e. a DLC of 8.

### 3.2 Reception and transmission of XCP packets

Upon reception of any XCP message the function

```
vuInt8 XcpPreCopy (PRECOPY_PARAM_TYPE PRECOPY_PARAM ) (5.3.1)
```

is called by the CAN-Driver and the XCP Packet is passed to the Protocol Layer by a call of the function:

```
void XcpCommand ( MEMORY_ROM vuInt32* pCommand )
```

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

```
void ApplXcpSend (vuInt8 len, MEMORY_ROM BYTEPTR msg ) (5.2.1)
```

and the XCP message is transmitted by the CAN-Driver service

```
vuInt8 CanTransmit ( CanTransmitHandle tmtHandle )
```

the successful transmission is confirmed by the CAN-Driver by a call of

```
void XcpConfirmation ( CanTransmitHandle tmtObject ) (5.3.2)
```

The confirmation is passed to the Protocol Layer by a call of

```
void XcpSendCallback (void )
```

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

### 3.3 Support of multiple CAN channels

Multiple CAN channels are supported by the XCP Transport Layer for CAN. However it is not possible to have multiple connections at one time. I.e. only one connection on one CAN channel is allowed.

The option 'multi connection protection' ensures that only one XCP Master communicates with the XCP Slave at one time.







## 4 Integration into the application






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

### 4.1 Files

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

Files of the XCP on CAN Transport Layer		
<code>xcp_can.c</code>	XCP on CAN Transport Layer. This file <b>must not</b> be changed by the user!	
<code>xcp_can.h</code>	API of the XCP on CAN Transport Layer. This file <b>must not</b> be changed by the application! This file has to be included prior to <code>XcpProf.h</code> .	
<code>v_def.h</code>	General Vector definitions of memory qualifiers and types. This file <b>must not</b> be changed by the application!.	
<code>_xcp.cfg</code>	XCP user config file template for the configuration on XCP Transport Layer for CAN with CANgen.	

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

Files generated by GENy		
<code>xcp_cfg.h</code>	Configuration file for XCP on CAN.	
<code>xcp_par.c</code>	Parameter definition for the XCP on CAN.	
<code>xcp_par.h</code>	External declarations for the parameters.	
<code>v_cfg.h</code>	General Vector configuration file for platform specifics.	
<code>v_inc.h</code>	General header for including the Vector CANbedded stack headers.	

Note that all files of XCP on CAN must not be changed manually except if CANgen is used for the configuration of the CAN-Driver. In this case only the generated files `xcp_cfg.h`, `xcp_par.c` and `xcp_par.h` are relevant and the additional general header for including the generated header files `_v_inc.h` has to be customized and renamed to `v_inc.h`.

### 4.2 Version changes

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

### 4.3 Integration of XCP on CAN into the application

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

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

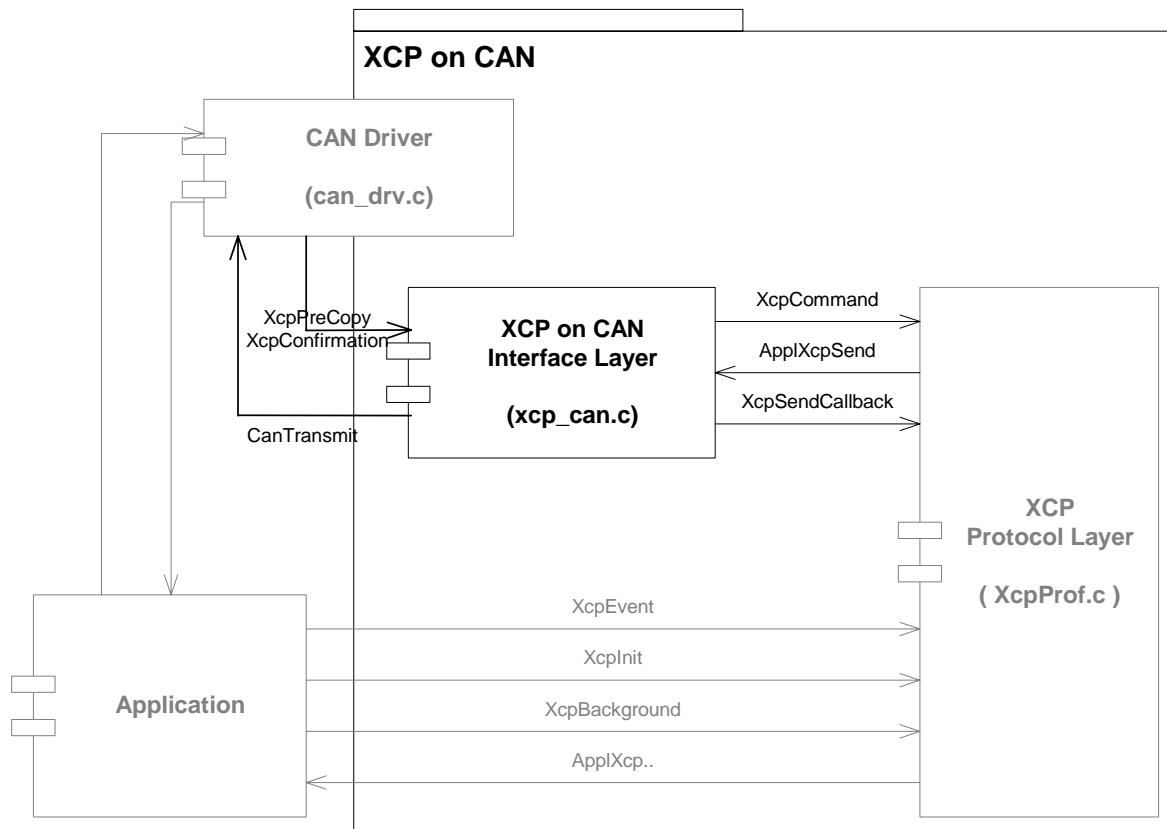


figure 4-1 Integration of XCP on CAN into the application



#### Practical Procedure

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

1. Configure XCP on CAN in the generation tool GENy and generate.
2. Include the include header file `v_inc.h` into all modules that access the XCP on CAN services or provide services that XCP on CAN uses.
3. Add all source files and generated source files in the make file and link it together with the data link layer and the application.
4. Initialize the data link layer after each reset during start-up before initializing XCP on CAN (interrupts have to be disabled until the complete initialization procedure is done) by calling `XcpInit`.

5. If required call the background function `XcpBackground` cyclically.
6. Integrate the desired XCP on CAN services into your application. Call especially the function `XcpEvent(channel)` cyclic with the appropriate cycle time and channel number.

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

## 5 Description of the API

The XCP on CAN 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 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 Transportation Layer for CAN is provided by three BCD coded constants:

```
V_MEMROM0 MEMORY_ROM vuint8 kXcpOnCanMainVersion =  
    (vuint8)(CP_XCPONCAN_VERSION >> 8);  
V_MEMROM0 MEMORY_ROM vuint8 kXcpOnCanSubVersion  =  
    (vuint8)(CP_XCPONCAN_VERSION);  
V_MEMROM0 MEMORY_ROM vuint8 kXcpOnCanReleaseVersion =  
    (vuint8)(CP_XCPONCAN_RELEASE_VERSION);
```



#### Example

Version 1.00.00 is registered as:

```
kXcpOnCanMainVersion    = 0x01;  
kXcpOnCanSubVersion     = 0x00;  
kXcpOnCanReleaseVersion = 0x00;
```

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

## **5.2 XCP Transport Layer for CAN services called by the Protocol Layer**

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

### **5.2.1 ApplXcpSend: Transmission of XCP Packets**

**ApplXcpSend**

Parameter	
-	-
Return code	
-	-
Functional Description	
Initialization of the XCP Transport Layer for CAN.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Not reentrant</li> <li>■ Call context of XcpInit</li> </ul>	

### 5.2.3 ApplXcpBackground: Background task of XCP Transport Layer for CAN

#### ApplXcpBackground

Prototype	
Single Channel	
Single Receive Channel	<code>void ApplXcpBackground ( void )</code>
Single Receive Buffer	N/a
Multiple Receive Buffer	N/a
Multi Channel	
Indexed (MRC)	Not supported
Code replicated (SRB)	N/a
Code replicated (MRB)	N/a
Parameter	
-	-
Return code	
-	-
Functional Description	
Cyclic background task of the XCP Transport Layer for CAN.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Not reentrant</li> <li>■ Call context of XcpBackground</li> </ul>	

### 5.3 XCP Transport Layer for CAN services called by the CAN-Driver

The following XCP Transport Layer for CAN functions are called by the CAN-Driver.  
The API of these functions can be found in the header of the CAN-Driver parameter file.

#### 5.3.1 XcpPreCopy: XCP message precopy function

##### XcpPreCopy

Prototype	
Single Channel CAN-Driver	
Single Receive Channel	<code>vuint8 XcpPreCopy ( CanRxInfoStructPtr rxStruct )</code>
Single Receive Buffer	<code>vuint8 XcpPreCopy ( CanReceiveHandle rxObject )</code>
Multiple Receive Buffer	<code>vuint8 XcpPreCopy ( CanChipDataPtr rxDataPtr )</code>
Multi Channel CAN-Driver	
Indexed (MRC)	<code>vuint8 XcpPreCopy ( CanRxInfoStructPtr rxStruct )</code>
Code replicated (SRB)	<code>vuint8 XcpPreCopy ( CanReceiveHandle rxObject )</code>
Code replicated (MRB)	<code>vuint8 XcpPreCopy ( CanChipDataPtr rxDataPtr )</code>
Parameter	
rxStruct	Pointer to RxInfoStruct
rxObject	Handle of the received object
rxDataPtr	Pointer to received data
Return code	
vuint8	kCanNoCopyData : no data needs to be copied by the CAN-Driver
Functional Description	
Precopy function that is called upon every reception of a XCP message.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Not reentrant</li> <li>■ Call context of CAN-Driver</li> </ul>	

### 5.3.2 XcpConfirmation: XCP message confirmation

#### XcpConfirmation

Prototype	
Single Channel CAN-Driver	
Single Receive Channel	<code>void XcpConfirmation ( CanTransmitHandle tmtObject )</code>
Single Receive Buffer	<code>void XcpConfirmation ( CanTransmitHandle tmtObject )</code>
Multiple Receive Buffer	<code>void XcpConfirmation ( CanTransmitHandle tmtObject )</code>
Multi Channel CAN-Driver	
Indexed (MRC)	<code>void XcpConfirmation ( CanTransmitHandle tmtObject )</code>
Code replicated (SRB)	<code>void XcpConfirmation ( CanTransmitHandle tmtObject )</code>
Code replicated (MRB)	<code>void XcpConfirmation ( CanTransmitHandle tmtObject )</code>
Parameter	
tmtObject	Transmit Handle of the confirmed message.
Return code	
-	-
Functional Description	
Confirmation function for the XCP message. This function is called by the CAN-Driver whenever the XCP message has been transmitted successful.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Not reentrant</li> <li>■ Call context of CAN-Driver</li> </ul>	

### 5.4 XCP Protocol Layer services called by the Transport Layer for CAN

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

- `void ApplXcpInterruptEnable(void)`
- `void ApplXcpInterruptDisable(void)`
- `void XcpCommand( MEMORY_ROM vuint32* pCommand )`
- `void XcpSendCallBack( void )`
- `vuint8 XcpGetState( void )`

For a description of the API and the functionality of these functions please refer to the Technical Reference XCP Protocol Layer [4].

### 5.5 CAN-Driver services called by the Transport Layer for CAN

The following CAN-Driver services are called by the Transport Layer for CAN:

- `vuint8 CanTransmit ( CanTransmitHandle tmtHandle )`

For a description of the API and the functionality of these functions please refer to the Technical Reference CAN Driver [5].



## 6 Configuration of XCP on CAN

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

Therefore if the CAN-Driver is configured with CANgen two generation tools are used:

- GENy for the configuration of the XCP Protocol Layer
- CANgen for the configuration of the CAN-Driver. The XCP Transport Layer for CAN has to be configured manually (refer to chapter 6.2).

### 6.1 Configuration of XCP on CAN with GENy

If GENy is used for the configuration of the whole CANbedded stack the configuration of XCP on CAN is conveniently done by GENy. No database attributes are required for the configuration of XCP on CAN.

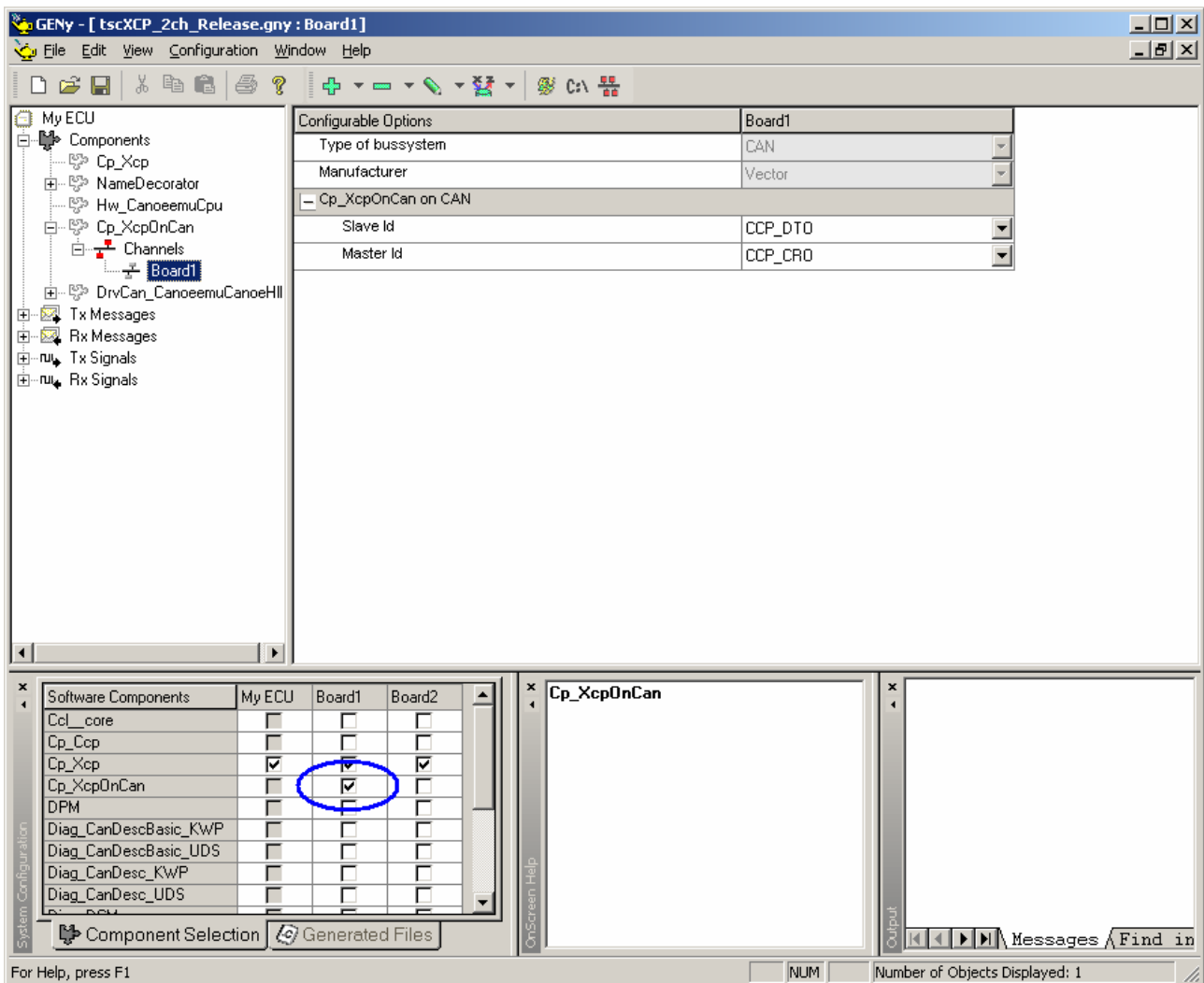


figure 6-1 Component selection

In order to configure the XCP Transport Layer for CAN (Cp\_XcpOnCan) it has to be activated on the designated channels. The activation of the XCP Transport Layer for CAN requires to activate the XCP Protocol Layer (Cp\_Xcp).

The configuration of each component is done on separate pages. Furthermore XCP Transport Layer for CAN has ECU specific and channel specific settings that have to be customized separately.

### 6.1.1 Main configuration page

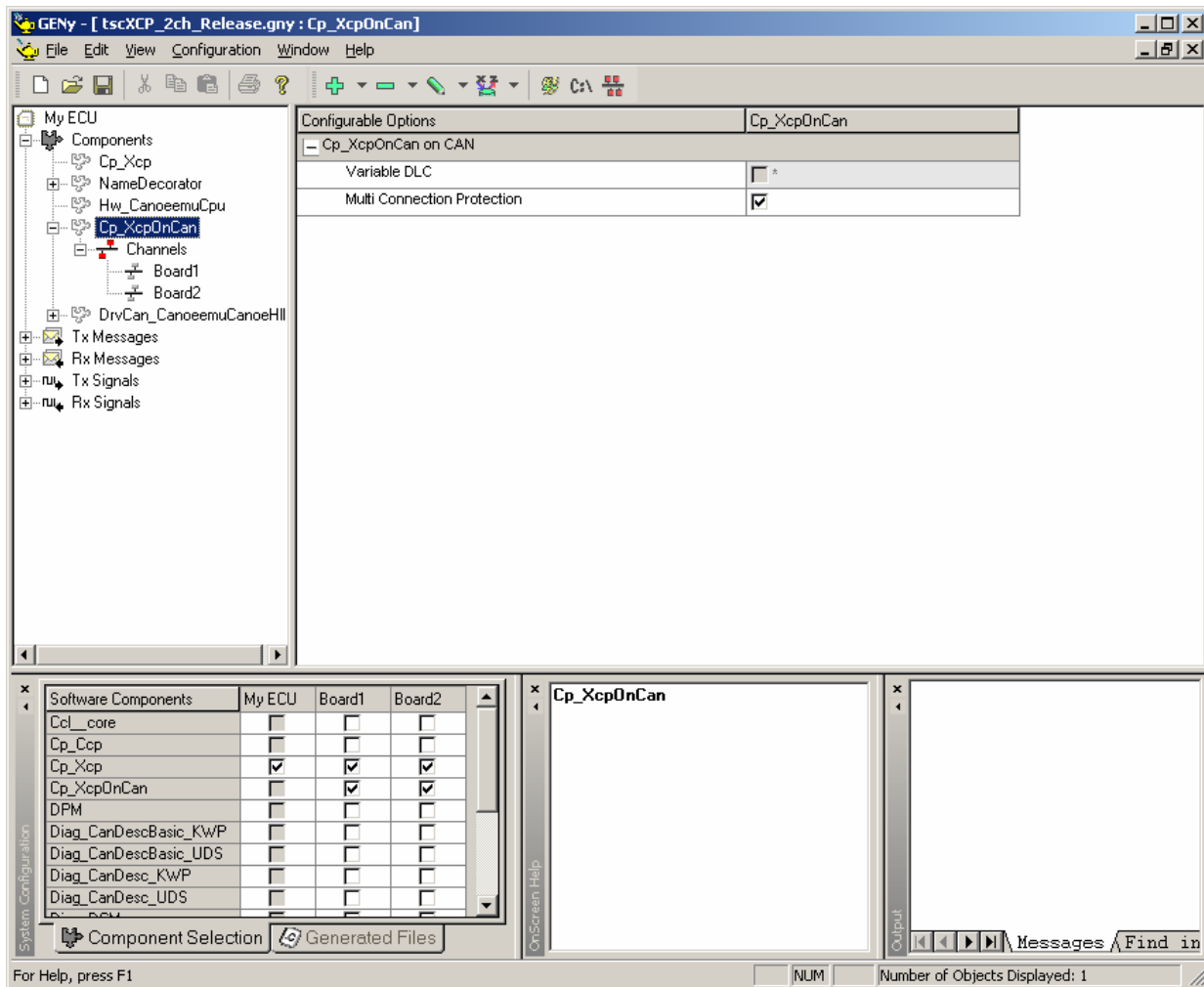


figure 6-2 Main configuration page of XCP on CAN Transport Layer

Configuration options	Value	Description
<b>XCP on CAN Transport Layer options</b>		
Variable DLC	<ul style="list-style-type: none"> <li>■ Enable</li> <li>■ Disable</li> </ul>	Activate/Deactivate the transmission of messages with variable DLC. This option is not available yet!
Multi connection protection	<ul style="list-style-type: none"> <li>■ Enable</li> <li>■ Disable</li> </ul>	Activate/Deactivate the protection against multiple connections. Only available if XCP on CAN is used on multiple CAN channels.

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

### 6.1.2 Channel configuration page

The messages can be selected in GENy within Component Cp\_XcpOnCan under the channel view (see below). The list boxes of the master/slave ID entry fields provide messages which fit to the requirements. The user can select the appropriate message.

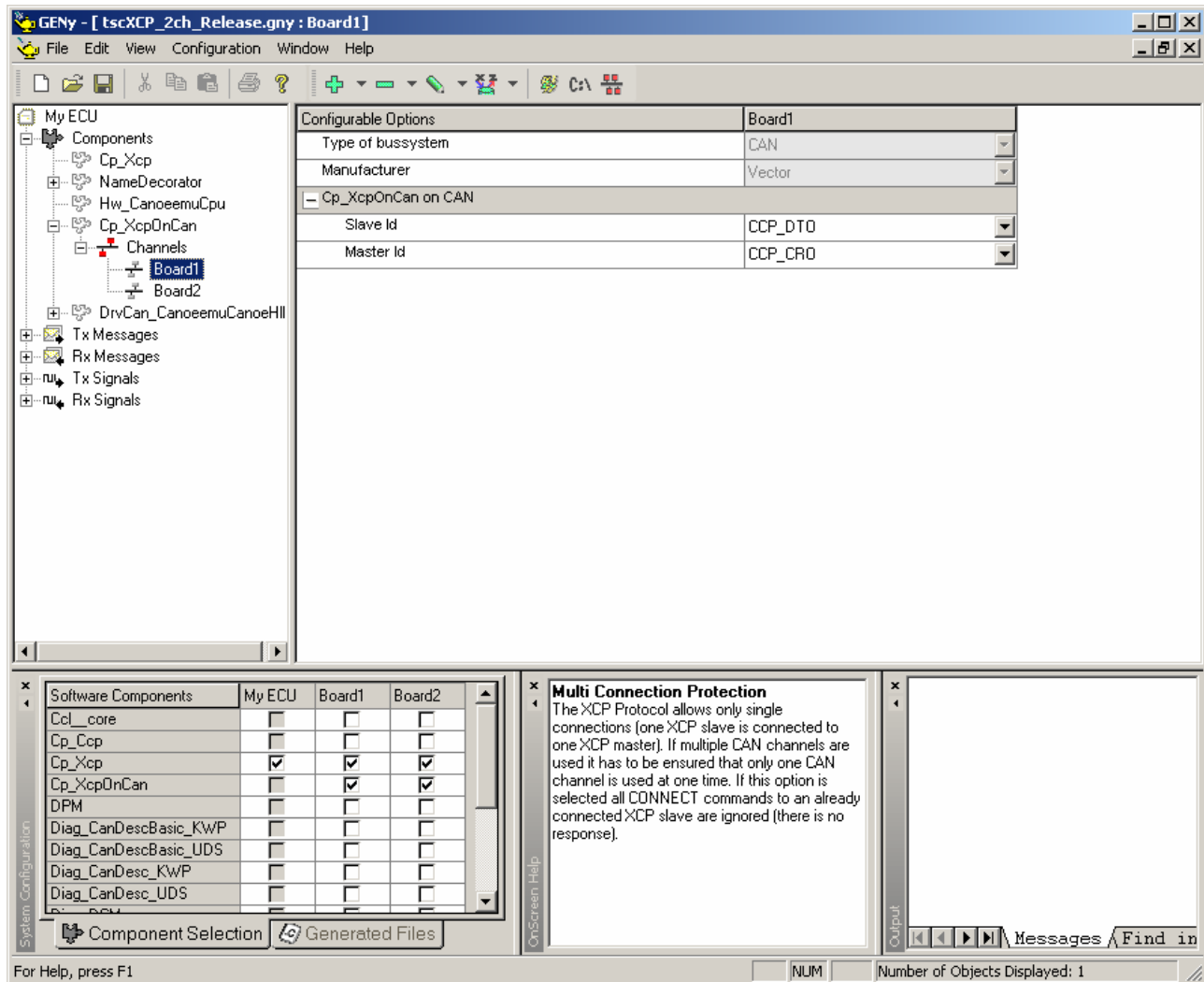


figure 6-3 Channel configuration page of XCP on CAN Transport Layer

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	XCP Slave Identifier This is the ID for Response Packets and DAQ packets. Only IDs that are not IL, TP or Diag messages and that have a DLC of 8 can be selected.
Master Id	RX-ID	XCP Master Identifier This is the ID for Command Packets and STIM Packets.

		Only IDs that are not IL, TP or Diag messages and that have a DLC of 8 can be selected.
--	--	---

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

In case there are no messages of type Application available, no selection can be made in the Master/Slave ID field.

In this case two possible solutions exist:

1. Change the attributes of the XCP message(s) in the database. There are special attributes to specify a message as NM, TP, IL, etc.

In case this is not possible, use solution 2:

2. Within GENy an IL message can be configured to be 'Appl' message. In Tx and Rx message view, set the checkbox 'Application Message' for the appropriate message (as shown below, message B1\_CCP). Then the type changes to Message Class 'Appl'. Now this message should be found in the slave or master ID selection.

	Generate	Channel	ID	Extended ID	Length [byte]	Application Message	Message Class
NM_Asr_Board1	<input checked="" type="checkbox"/> *	*	0x51c	<input type="checkbox"/> *	8	<input type="checkbox"/>	NM
B1_CCP	<input checked="" type="checkbox"/> *	*	0x791	<input type="checkbox"/> *	8	<input checked="" type="checkbox"/>	APPL
ApplMM_Board1	<input checked="" type="checkbox"/> *	*	0x78a	<input type="checkbox"/> *	8	<input type="checkbox"/>	IL

figure 6-4 Changing message attribute to "Application Message"

### 6.1.3 Multiple Identity configuration

For information about setting up a Multiple Identity configuration please refer to the according Technical Reference. This chapter only explains the XCP specific configuration.

For each configured Identity the Slave Id and Master Id must be configured. For a configuration with Multiple Identity this is not done on the channel dependent page as shown in chapter 6.1.2, but on the Identities page below the channel dependent page (s. figure below).

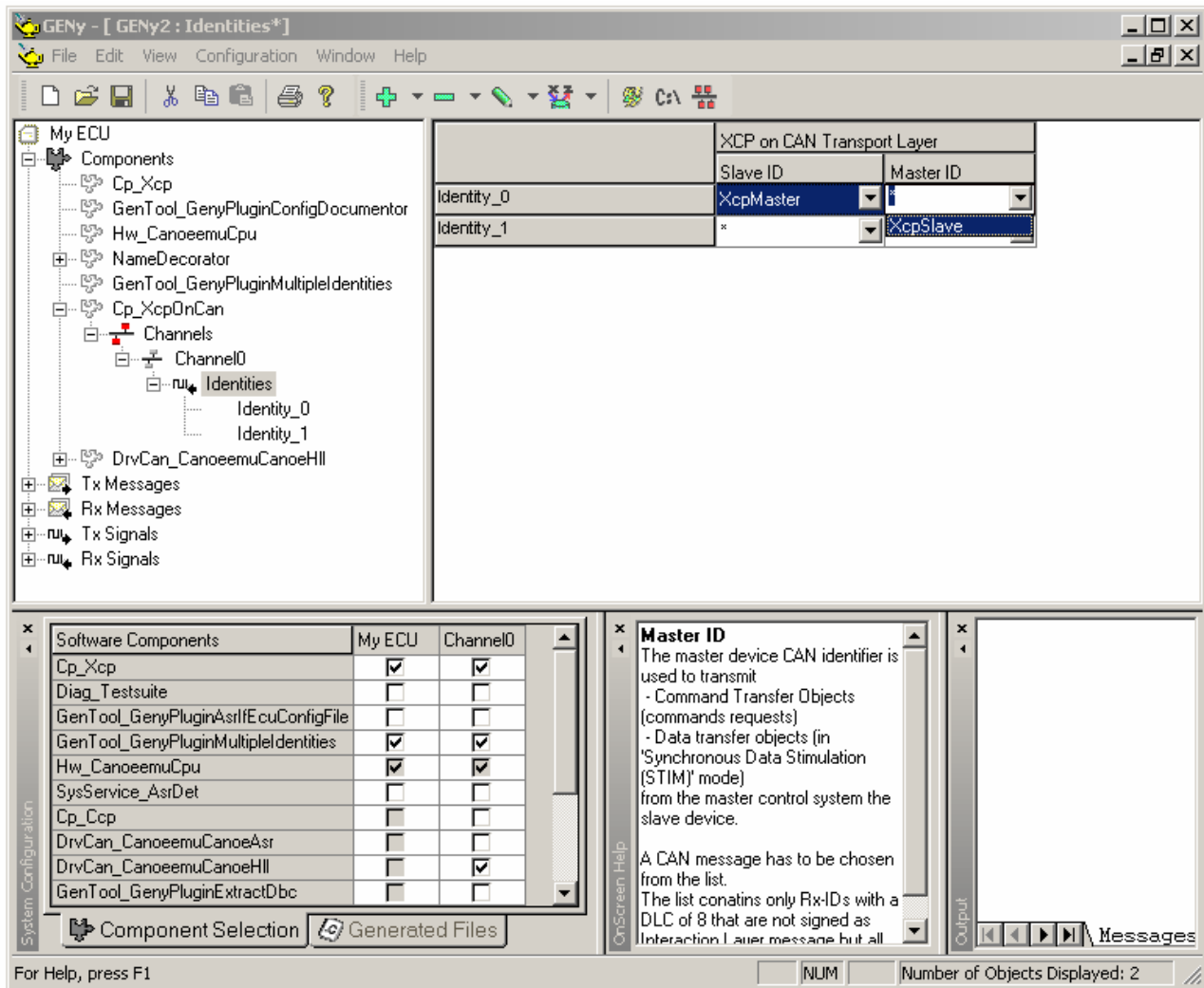


figure 6-5 Channel configuration page for Multiple Identity configurations of XCP on CAN Transport Layer

## 6.2 Configuration of XCP on CAN with GENy and CANGen

If the CAN-Driver is configured by CANGen the configuration of the XCP Transport Layer for CAN has to be done manually. The configuration of the XCP Protocol Layer is performed conveniently by GENy.

Reference the user config file `xcp.cfg` on the XCP Protocol Layers main page in GENy as shown in figure 6-6 and customize it according to Table 6-3.

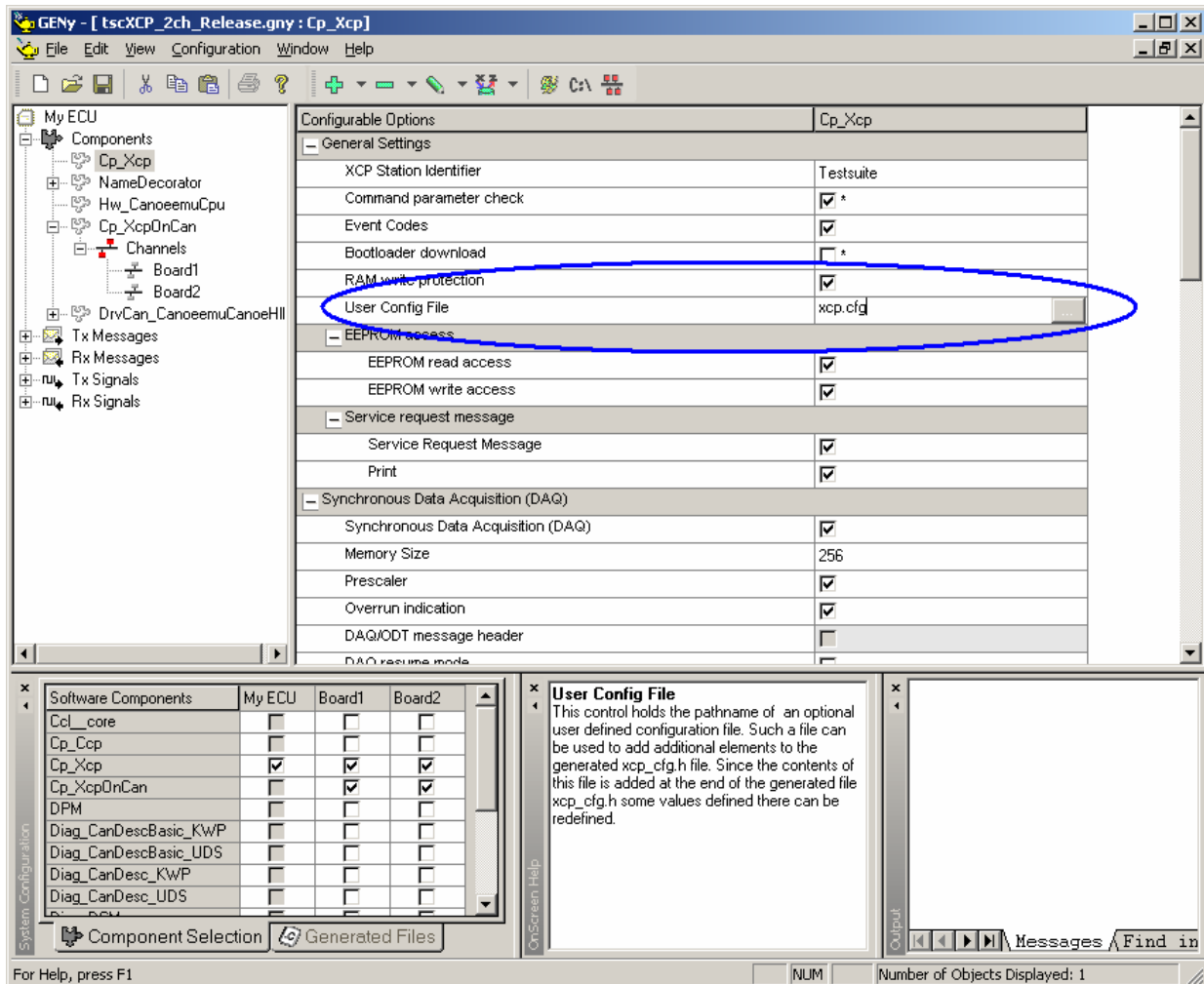


figure 6-6 Adding a user config file in GENy

Configuration options	Value	Description
XCP on CAN Transport Layer Options		
XCP_TRANSPORT_LAYER_TYPE_CAN		Activate the XCP on CAN Transport Layer
XCP_xxx_VARIABLE_DLC	ENABLE <b>DISABLE</b>	Activate/Deactivate the transmission of messages with variable DLC. This option is not available yet!
XCP_xxx_MULTI_CHANNEL	<b>ENABLE</b> DISABLE	Enable support of multiple CAN channels
XCP_xxx_MULTI_CONNECTION_PROTECTION	ENABLE <b>DISABLE</b>	Protection against multiple connections. Only available if XCP on CAN is used on multiple CAN channels
kXcpNumberOfCanChannels	2..255	Specify the number of CAN channels that use XCP on CAN. Only available if XCP on CAN is used on multiple CAN channels

Table 6-3 Options and configuration of XCP on CAN Transport Layer

The XCP Slave ID and XCP Master ID have to be configured in the generation tool that configures the CAN-Driver. If XCP on CAN is only used on one CAN channel refer to chapter 6.2.1 'XCP on CAN uses only one CAN channel' otherwise refer to chapter 6.2.2 'XCP on CAN uses multiple CAN channels'.

### 6.2.1 XCP on CAN uses only one CAN channel

The generation tool CANgen has to be configured as follows:

- Add the precopy function `XcpPreCopy` for the XCP Master ID in your generation tool.
- Add the confirmation function `XcpConfirmation` for the XCP Slave ID in your generation tool.

The Transmit handle and the data buffer of the XCP Slave ID have to be defined in the user config file `xcp.cfg`:

- `#define XcpGetTransmitHandle() TransmitHandleOfSlaveId`  
`#define XcpGetTransmitDataPtr() TransmitDataBufferOfSlaveId`
- Replace `TransmitHandleOfSlaveId` by the transmit handle of the XCP Slave Id that can be found in the header `<node>.h`.
- Replace `TransmitDataBufferOfSlaveId` by the transmit data buffer of the XCP Slave Id that can be found in the header `<node>.c`.



#### Example for Slave Id XCP\_DTO

```
#define XcpGetTransmitHandle() CanTxXCP_DTO
#define XcpGetTransmitDataPtr() (XCP_DTO._c)
```

### 6.2.2 XCP on CAN uses multiple CAN channels

If XCP on CAN is used on multiple CAN channels (e.g. vehicle bus and private CAN) the configuration of the XCP Master ID and XCP Slave ID is done by the following steps.

The generation tool CANGen has to be configured as follows:

- Add the precopy function `XcpPreCopy` for the XCP Master IDs in your generation tool.
- Add the confirmation function `XcpConfirmation` for the XCP Slave IDs in your generation tool.

The Transmit handles and the data buffers of the XCP Slave IDs have to be defined in an additional source file.

```
V_MEMROM0 V_MEMROM1 CanTransmitHandle V_MEMROM2 xcpTxHandleField[] =
{
    Tx handle of XCP Slave ID 1,
    Tx handle of XCP Slave ID 2
};

V_MEMROM0 V_MEMROM1 TxDataPtr V_MEMROM2 xcpTxDataPtrField[] =
{
    data bufferTx of XCP Slave ID 1,
    data bufferTx of XCP Slave ID 2
};
```

The following macros have to be defined in the user config file `xcp.cfg`:

- `#define XcpGetTransmitHandle() (xcpTxHandleField[xcpChannelNumber])`
- `#define XcpGetTransmitDataPtr() (xcpTxDataPtrField[xcpChannelNumber])`



#### Example for Slave Id XCP\_DTO and XcpSlave

User config file `xcp.cfg` :

```
#define XcpGetTransmitHandle() (xcpTxHandleField[xcpChannelNumber])
#define XcpGetTransmitDataPtr() (xcpTxDataPtrField[xcpChannelNumber])
```

C Source Code:

```
V_MEMROM0 V_MEMROM1 CanTransmitHandle V_MEMROM2 xcpTxHandleField[] =
{
    CanTxXCP_DTO,
    CanTxXcpSlave
};

V_MEMROM0 V_MEMROM1 TxDataPtr V_MEMROM2 xcpTxDataPtrField[] =
{
    XCP_DTO._c,
    XcpSlave._c
};
```



## 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 Channel API

XCP on CAN is only available with a single channel API.

However all currently available single and multiple channel APIs of the CAN-Driver are supported.

### 7.1.5 Multiple Identity only supported for single channel configuration

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

## 8 FAQ

### 8.1 Transmit queue of CAN-Driver is disabled



#### FAQ

How to operate XCP on CAN if the transmit queue of the CAN-Driver is disabled.

If the transmit queue of the CAN-Driver 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 `ApplXcpBackground()` 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](http://www.vector-informatik.com)**