

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

HiCOCAN-SW

Software Documentation for Windows

Rev17 / 10.08.2012

emtrion GmbH

© Copyright 2012 **emtrion GmbH**

All rights reserved. This documentation may not be photocopied or recorded on any electronic media without written approval. The information contained in this documentation is subject to change without prior notice. We assume no liability for erroneous information or its consequences. Trademarks used from other companies refer exclusively to the products of those companies.

Revision: **17 / 10.08.2012**

Rev	Date/Signature	Changes
1	24.06.1999/Ft	Initial revision
2	15.12.1999/Ft	Chapter Windows CE driver and Multitasking added
5	28.07.2000/Ft	Driver section modified
6	08.05.2001/Rr	HiCOCAN-PCI added
7	15.05.2002/Ny	Chapter Linux added
8	26.09.2002/Rr	HiCOCAN-CPCI added
9	31.01.2003/Ft	Corrections, added HiCOCAN-104-2L, HiCOCAN-104-2H and new API functions
10	30.04.2004/Rr	Chapter Linux moved to a separate document
11	23.05.2007/Rr	Software support for HiCOCAN-MiniPCI added
12	24.04.2008/Rr	Reorganizing all manuals
13	10.07.2008/Rr	HiCOCAN-PCI104 added
16	20.12.2010/Bi	HiCOCAN-Mini-PCI with 4 channels added
17	10.08.2012/Ft	Information about Windows x64-Versions added Review of the complete manual

Inhaltsverzeichnis

1 Introduction 5
1.1 Software Package Contents 6
1.2 Overview of the API Functions 8
2 Using HiCOCAN on 64-bit Windows (Windows x64) 10
2.1 Naming of the driver DLL 10
2.2 Installation of the package 10
2.3 Configuration Tool and CAN Monitor on Windows x64 10
2.4 Using 32-bit applications under Windows x64 10
2.5 Creating 64-bit applications 10
3 Application Interface 11
3.1 Basic Structure of an Application 11
3.1.1 Data Structures 11
3.1.2 Return Values of the Functions 13
3.2 Numbering the CAN Nodes 15
3.3 Communication with HiCOCAN 16
3.3.1 Command to the Board 16
3.3.2 Determining the Status Information 16
3.3.3 Message Transfer with the Windows Driver 17
3.4 Interrupt Handling 17
3.4.1 Interrupt Handling in the Windows Driver 17
3.5 Multitasking 18
3.6 General Functions 19
3.6.1 HiCOCANOpenDriver() 19
3.6.2 HiCOCANCloseDriver() 19
3.6.3 HiCOCANResetDriver() 20
3.6.4 HiCOCANGetErrorString() 20
3.6.5 HiCOCANGetDriverInformation() 21
3.7 Functions for Controlling the CAN Nodes 22
3.7.1 HiCOCANOpen() 22
3.7.2 HiCOCANClose() 22
3.7.3 HiCOCANStart() 23
3.7.4 HiCOCANStop() 24
3.7.5 HiCOCANReset() 24
3.7.6 HiCOCANResetContr() 25
3.7.7 HiCOCANClrOverrun() 26
3.7.8 HiCOCANAbortTransmit() 27
3.7.9 HiCOCANRegisterEvent() 27
3.7.10 HiCOCANCanCtrl() 30
3.8 Timestamp 32
3.8.1 HiCOCANSetTimestamp() 32
3.9 Read/Write Functions 33
3.9.1 HiCOCANWrite() 33
3.9.2 HiCOCANRead() 34
3.9.3 HiCOCANReadEx() 35

3.10 Status Request 37
3.10.1 HiCOCANState() 37
3.10.2 HiCOCANStateTrans() 39
3.10.3 HiCOCANStateContr() 40
3.10.4 HiCOCANTraQState(), HiCOCANRecQState() 41
3.11 Modifying the Communication Parameters 43
3.11.1 HiCOCANSetAcceptMask() 43
3.11.2 HiCOCANSetTimingReg(), HiCOCANSetTimingRegEx(), HiCOCANSetBaud() 44
3.11.3 HiCOCANParameter() 46
4 Configuration Tool under Windows 48
4.1 The Menus and Buttons 49
4.1.1 File Menu 49
4.1.2 Boards Menu 50
4.1.3 Help Menu 52
4.1.4 The Symbol Bar 53
4.2 Format of the Configuration Files 53
4.2.1 The HiCOCAN Section 53
4.2.2 The Sections CAN-NODE1 and CAN-NODE2 53
5 Sample Application 55
5.1 Detailed Description 55
6 Troubleshooting (HiCOCAN-xxx) 57
6.1 Support 57
7 Reference 58

1 Introduction

HiCOCAN is the perfect solution for connecting your PC to a CAN net. No matter what applications you wish to develop, HiCOCAN provides the performance required for your specific design needs, at the highest bus rates.

The following HiCOCAN variants are available:

Name	Description
HiCOCAN-PCI-1	Standard PC card for the PCI bus of a standard PC with one CAN interface
HiCOCAN-PCI-2	Standard PC card for the PCI bus of a standard PC with two CAN interfaces
HiCOCAN-CPCI	Compact PCI card with two CAN interfaces
HiCOCAN-MiniPCI	miniPCI type IIIA board with two CAN interfaces
HiCOCAN-MiniPCI-4CH	PC/104 variant with two CAN interface, high-speed (jumpers)
HiCOCAN-104-2H	PC/104 variant with two CAN interface, high-speed (jumpers)
HiCOCAN-104-2L	PC/104 variant with two CAN interface, low-speed, fault-tolerant (jumpers)
HiCOCAN-PCI104-2H	PCI104 variant with two CAN interface, high-speed (jumpers)
HiCOCAN-PCI104-2L	PCI104 variant with two CAN interface, low-speed, fault-tolerant (jumpers)

All boards are shipped with a preinstalled firmware, which handles Layer 2 of the ISO/OSI reference model. The PC's function libraries are used for communications with the firmware and so provide optimum access to the CAN net.

This manual provides a description of the software, where no distinction is made between all the different types of CAN boards.

1.1 Software Package Contents

The delivery of HiCOCAN-SW includes the following:

- Various drivers:

HiCOCAN	PCI	CPCI	PCI 104	MiniPCI (-2CH)	MiniPCI -4CH	104-2H	104-2L
Windows 98	X	X	X	X		X	X
Windows ME	X	X	X	X		X	X
Windows NT	X	X	X	X		X	X
Windows 2000	X	X	X	X		X	X
Windows XP (32bit)	X/ X	X/ X	X	X/ X		X	X
Windows Vista (32bit)	X	X		X	X		
Windows 7 (32bit)	X	X		X	X		
Windows 7 (64bit)	X	X	X	X	X		
Windows CE						Available for SH4/SH3, ask emtrion for specific CPU support	

The red Xs mark the support of the most recent driver (Version 6.x). Note that with this driver only HiCOCAN-PCI and MiniPCI cards are supported. Also only the OSs Windows XP, Vista and 7 are supported.

- CONFIG.EXE configuration tool
- CAN monitor (Sample application)
- Libraries and header files for the application development
- Sample application
- Documentation

Directory				Content
DevFiles				
	DemoApp			
		hicocandemo		Sources of the sample application for boards with two channels
			Release	Executable file
		Hicocandemo_4ch		Sources of the sample application for HiCOCAN-MiniPCI boards with four channels
			Release	Executable file
	DriverLibs			
		HICOCAN_MiniPCI		Libraries and Header files for developing Windows applications for HiCOCAN-MiniPCI (-2CH/-4CH) boards
		HICOCAN_PCI104		Libraries and Header files for developing Windows applications for HiCOCAN-PCI104 boards
		HICOCANPCI_CPCI		Libraries and Header files for developing Windows applications for HiCOCAN-PCI and -CPCI boards
Manuals				The manuals for installation, API description (this manual) and technical specifications
Tools				Configuration Tool and CAN monitor
	Firmware			
		HICOCAN		Firmware for the HiCOCAN-104/HiCOCAN-PCI and HiCOCAN-CPCI board
		HiCOCANMiniPCI2ch		Firmware for the HiCOCAN-MiniPCI-2ch board
		HiCOCANMiniPCI4ch		Firmware for the HiCOCAN-MiniPCI-4ch board

1.2 Overview of the API Functions

The APIs of the drivers (on CD) for the various operating systems slightly differ from each other. This is due to different versions of the individual drivers. For example, the latest drivers include additional functionality.

The table below provides an overview of the windows drivers and their API functions.

If you require an API function in a driver that does not yet support this functionality, please contact us.

Remark: In the table below the driver for Windows 9x / Me / NT4 / 2000 and the 32-bit versions of Windows XP / Vista and 7 is called Win32. The driver for the 64-bit version of Windows 7 is called x64.

Function	Drivers for … boards under …			
	PC104		PCI/CPCI/PCI104 MiniPCI (-2CH/-4CH)	
	Win32	WinCE	Win32	x64
General Functions				
HiCOCANOpenDriver	X	X	X	X
HiCOCANCloseDriver	X	X	X	X
HiCOCANResetDriver	X	X	X	X
HiCOCANGetErrorString	X	X	X	X
HiCOCANGetExtendedErrorString	X	X	X	X
HiCOCANSetResource				
HiCOCANGetDriverInformation	X	X	X	X
Functions for Controlling the CAN nodes				
HiCOCANCanCtrl		X	X	X
HiCOCANOpen	X	X	X	X
HiCOCANStart	X	X	X	X
HiCOCANStop	X	X	X	X
HiCOCANReset	X	X	X	X
HiCOCANResetContr	X	X	X	X
HiCOCANClrOverrun	X	X	X	X
HiCOCANAbortTransmit	X	X	X	X
HiCOCANRegisterEvent	X	X	X	X
Timestamp				
HiCOCANSetTimestamp	X	X	X[2)]	X[2)]
Read/Write Functions				
HiCOCANWrite	X	X	X	X
HiCOCANRead	X	X	X	X
HiCOCANReadEx	X	X	X	X
Status Request				
HiCOCANState	X	X	X	X
HiCOCANState			X[4)]	X[4)]
HiCOCANStateContr	X	X	X	X
HiCOCANTraQState	X	X	X	X
HiCOCANRecQState	X	X	X	X

Function	Drivers for … boards under …			
	PC104		PCI/CPCI/PCI104 MiniPCI (-2CH/-4CH)	
	Win32	WinCE	Win32	x64
Modifing the communication Parameters				
HiCOCANSetAcceptMask	X	X	X[1)]	X[1)]
HiCOCANSetBaud	X	X	X	X
HiCOCANSetTimingReg	X	X	X[1)]	X[1)]
HiCOCANSetTimingRegEx			X[3)]	X[3)]
HiCOCANParameter	X	X	X[1)]	X[1)]

[1)] Not supported by HiCOCAN-MiniPCI

[2)] Supported by HiCOCAN-MiniPCI with limitations

[3)] Only supported by HiCOCAN-MiniPCI

[4)] Only supported by HiCOCAN-PCI104 with fault tolerant transceiver

2 Using HiCOCAN on 64-bit Windows (Windows x64)

Starting with revision 6.1 the Windows package supports in addition to the 32-bit Editions of Windows XP, Vista and Windows 7 also the 64-bit Editions of Windows 7.

2.1 Naming of the driver DLL

The name of the driver DLL are extended for the 64-bit driver compared with name of the 32-bit driver. The extension is that the DLL name of the 64-bit driver contains a part _x64. The following table shows an overview about the available driver DLLs and their names:

Board type	32-bit OS	64-bit OS
HiCOCAN-PCI / HiCOCAN-CPCI	HiCanPci.DLL	HiCanPci_x64.DLL
HiCOCAN-MiniPci (-2CH/-4CH)	HiCanMiniPci.DLL	HiCanMiniPci_x64.DLL
HiCOCAN-PCI104	HiCanPci104.DLL	HiCanPci104_x64.DLL

2.2 Installation of the package

The installer package recognizes if it runs on a 32-bit or 64-bit edition of Windows and installs automatically the correct driver revision if the driver installation is selected during the installation. On the 64-bit editions of Windows the installer package copies automatically the 32-bit and 64-bit DLLs of the driver API into the system to the correct locations in the system. The installer copies also the 64-bit edition of the configuration tool and the CAN monitor on 64-bit systems.

2.3 Configuration Tool and CAN Monitor on Windows x64

As explained in the former section the installer package automatically installs the 64-bit editions of the Configuration Tool and the CAN monitor if the installation of this tools are selected and the installer recognizes a 64-bit edition of the Windows. In this case you have to select the 64-bit version of the DLLs, which are located in the %WINDIR%\system32 directory (%WINDIR% specifies the installation directory of your Windows operating system, typically C:\Windows)..

2.4 Using 32-bit applications under Windows x64

Existing 32-bit applications can use the HiCOCAN boards under Windows 7 x64 without any changes. During the installation of the package the installer places the 32-bit version of the DLLs in the %WINDIR%\SysWOW64 directory so that the DLLs are found automatically from the applications. This allows the communication through HiCOCAN boards.

2.5 Creating 64-bit applications

The application programming interface (API) is the same for 64-bit applications as for 32-bit applications. The only difference is in the linker settings. For 64-bit applications you have to use the libraries (.lib) which contains the part _x64 in their name (32-bit applications have to use the libraries without this part). Everything else is the same.

3 Application Interface

The transfer constants and return values mentioned in this chapter are defined in the supplied header file HiCOCAN.h.

3.1 Basic Structure of an Application

The basic structure of an application is shown by Fig. 1:

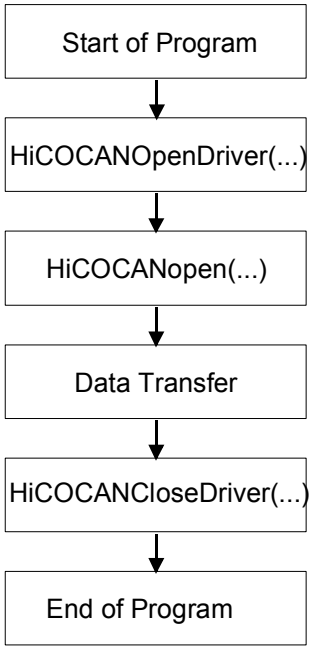


Fig. 1: Basic structure of an application

The first function **HiCOCANOpenDriver** executes all the necessary initializations within the software.

The **HiCOCANOpen** function must be executed for each CAN node. After this, the application is allowed to make use of all other software functions (DLL) without any restrictions.

If a CAN node was programmed with StartMode 0 for using the configuration tool ,the **HiCOCANStart** function must be called before the data traffic with the CAN net is started. For HiCOCAN-MiniPCI this command must be called anyway.

The application is closed with **HiCOCANCloseDriver**. The resources used by the software are then freed again. HiCOCANClose does not have to be called for each individual node, since this is done by the HiCOCANCloseDriver function.

3.1.1 Data Structures

In the **HiCOCAN.h** C header file two data structures are defined which are required for both reads and writes and for reading the timestamps:

```
typedef struct
{
     BYTE   ff;        /* frame format:   0 = basic CAN,        */
                       /*                 1 = extended CAN      */
     BYTE   rtr;       /* 0 = normal frame, 1 = remote frame    */
     BYTE   dlc;       /* data length 0..8                      */
     DWORD  id;        /* telegram ID                          */
     BYTE   data[8];   /* data                                  */
     sTS    timestamp; /* time stamp*/
}sPCCanMsg, *psPCCanMsg;
```

ff specifies whether the CAN message is a Basic CAN message with 11 identifier bits (HiCOCAN_FORMAT_BASIC) or a CAN message in the Extended CAN format with 29 identifier bits (HiCOCAN_FORMAT_EXTENDED).

rtr indicates the frame type; HiCOCAN_REMOTE_FRAME stands for remote messages and HiCOCAN_NORMAL_FRAME for normal messages

For HiCOCAN-MiniPCI there is an extension available. If bit 7 is set, this telegram has been received during an overrun condition of the CAN network. This means that some telegrams are possibly missing. If bit 6 is set, this telegram has been received with a fault tolerant error condition on the CAN network. This bit makes only sense if there is a fault tolerant transceiver available.

The number of data bytes is entered with **dlc**.

The ID of a message is always specified with the **id** field; no matter if it is a Basic CAN message or an Extended CAN message.

An array of bytes **data[8]** is reserved for the message data.

Each received message is provided with a timestamp:

```
typedef struct
{
     WORD   day;
     BYTE   hour;
     BYTE   min;
     BYTE   sec;
     WORD   ms;
     WORD   us;            /* micro seconds; max. resolution 10µs */
}sTS, *psTS;
```

In addition, the C header file contains the sHiCOCANDriverInfo data structure which is required for the HiCOCANGetDriverInformation function:

```
typedef struct
{
     BYTE   bStructVersion;
     LPBYTE lpbVersionStringASCII;
     LPWORD lpwVersionStringUnicode;
     DWORD  dwSubVersion;
     WORD   wSizeOfPCRecQ;
} sHiCOCANDriverInfo, *psHiCOCANDriverInfo;
```

The bStructVersion element specifies the structure's version number. It has to be initialized by the application (currently, to 1).

The lpbVersionStringASCII element contains a pointer to a buffer which is at least 30 bytes in size. In this buffer, an ASCII string is stored that contains the revision number as ASCII text. For driver version 4.5, it will look as follows: $ProjectRevision : 4.5$

The element lpwVersionStringUnicode contains a pointer to a buffer which is at least 60 bytes in size. In this buffer, a string is stored that contains the revision number as unicode text. It will look like the ASCII string.

The dwSubVersion element stores the version number of the kernel driver.

The wSizeOfPCRecQ element stores the size of the driver-internal receive queue.

3.1.2 Return Values of the Functions

All software functions return a 32-bit code as return value. The following is a list of all return values available:

Return value	Significance
HTX_SUCCESS	No error detected
HTX_BUSOFF	No error has occurred while the function is running; the accessed CAN node is in Bus-Off state
HTX_OVERRUN	No error has occurred while the function is running; the accessed CAN node is in Overrun state
HTX_CANERROR	No error has occurred while the function is running; however, the error counters of the accessed can node are unequal to 0.
HTX_ERROR	General error
HTX_ERROR_ALREADY_OPENED	CAN node already open
HTX_ERROR_ALREADY_USED	CAN node already used by another process

Return value	Significance
HTX_ERROR_APPLICATION	Invalid order of function calls through the application (A driver function was called without calling HiCOCANOpenDriver)
HTX_ERROR_BOARD_NOT_ RUNNING	Error in the bootstrap loader or the interrupt service routine cannot be installed
HTX_ERROR_CANNOT_SET_IRQ	Desired interrupt not available
HTX_ERROR_CHECKSUM_ BOOTSTRAPPER	Checksum error in the bootstrap loader area
HTX_ERROR_CHECKSUM_ FIRMWARE	Checksum error in the firmware area
HTX_ERROR_DRIVER	An error occurred in the driver. Possible causes: • Driver has not been installed properly • The HiCOCANOpenDriver function was not properly called by the application
HTX_ERROR_EMPTY_QUEUE HTX_ERROR_RECEIVEQUEUE_EMPTY	Receive queue of the specified CAN node is empty
HTX_ERROR_FULL_QUEUE HTX_TRANSMITQUEUE_FULL	Transmit queue of the specified CAN node is full
HTX_ERROR_ILLEGAL_ID	Illegal message ID
HTX_ERROR_ILLEGAL_LENGTH	Illegal message length
HTX_ERROR_MULTIPLE_NODE	Multiple CAN nodes in the system (e.g. if there are two modules with the same module number)
HTX_ERROR_NOCONFIG	The flash does not contain valid configuration data for the specified CAN node.
HTX_ERROR_NOT_SUPPORTED	Function not supported by the module's firmware.
HTX_ERROR_REGISTRY	An error occurred while accessing the Plug and Play configuration data in the Windows registry (the registry could not be read).
HTX_ERROR_RESOURCE	Error while accessing the Plug and Play configuration data; no resources were allocated to the module (addresses and interrupts)

Return value	Significance
HTX_ERROR_SYSTEM	The firmware reported a system error. Bits 16 through 23 of the retrun value contain an error number. If the error number equals 1, the HiCOCAN module is busy. Other error numbers are internal firmware errors. For further information, please contact the emtrion support team.
HTX_ERROR_TRIGGERLEVELSET_ INVALID	The value specified for the trigger level dwTriggerLevelSet is not valid, because the receive queue does not have this size.
HTX_ERROR_TRIGGERLEVELRESET_ INVALID	The value specified for the trigger level dwTriggerLevelReset is not valid, because it is greater or queal to the trigger level dwTriggerLevelSet.
HTX_ERROR_UNKNOWN_BOARD	The specified board could not be found. This return value is also output if the softwar was not informed about the CAN nodes present on the board with the HiCOCANOpen function.
HTX_ERROR_UNKNOWN_NODE	The software was unable to find the specified CAN node (with the HiCOCANOpen function). The error does not occur if CAN node 2 is to be accessed on a board where only one CAN node is installed. With all other software functions this return value means that the software (DLL) was not informed about the CAN node via HiCOCANOpen function.

3.2 Numbering the CAN Nodes

It is possible to operate up to 4 HiCOCAN boards (not HiCOCAN-MiniPCI) in a system; the boards must be given different board numbers.

The CAN node is selected via the node number (can) according to the following formula:

```
can = 2 * board number + number of the CAN controller - 1
```

"Number of the CAN controller" may have a value of 1 or 2, which results in the following values for can:

Number of CAN node	Location of the CAN node
0	Module 0, CAN controller 1
1	Module 0, CAN controller 2
2	Module 1, CAN controller 1
3	Module 1, CAN controller 2
4	Module 2, CAN controller 1
5	Module 2, CAN controller 2
6	Module 3, CAN controller 1
7	Module 3, CAN controller 2

Table 1: Assigning the CAN node number to the CAN controllers on max. 4 boards

Note
The numbering of the CAN nodes depends on the number of the CAN nodes actually available on the board.
The number of physically available CAN nodes, however, will not be checked!

HiCOCAN MiniPCI Boards
You can only operate 1 MiniPCI (2 or 4 channels) board at a time. So, in case of 2 channels the nodes are numbered 0 and 1. In case of 4 channels: 0, 1, 2, 3.

3.3 Communication with HiCOCAN

3.3.1 Command to the Board

The board and the installed CAN nodes are controlled via entries in specific command cells of the DPM. The firmware analyzes these cells as follows:

- First it is checked whether the command cell is still in use by a previous command.
- If the cell is free, the required data can be entered in the DPM.
- The information about the command to be executed by the board is entered in the command cell.
- The firmware executes the command and then resets the command cell.

3.3.2 Determining the Status Information

The status information contained in the communication areas in the DPM is updated each time the firmware is modified. It is thus sufficient to read out the corresponding state via the functions provided.

3.3.3 Message Transfer with the Windows Driver

Using the HiCOCANRead or the HiCOCANReadEx function, you are able to read the received messages from the HiCOCAN module's DPM. You can write the messages to be transmitted to the module's DPM with the HiCOCANWrite function.

All three functions have a timeout time. This timeout time takes effect when the receive queue is empty or the transmit queue is full, respectively. In this case, the functions HiCOCANRead, HiCOCANReadEx and HiCOCANWrite are waiting for a free entry in the transmit queue until the timeout time has elapsed at the latest, before returning an error. This is done via driver-internal events by means of which the corresponding functions are blocked until the event occurs or the timeout time has elapsed. The use of those events prevents the functions from wasting runtime.

The events are set to the "Signalled state" via the interrupt service routine. The routine first processes the following causes of an interrupt:

- The new message was entered in the DPM receive queue.
- The firmware has taken a message from the DPM transmit queue.
- At least one message could not be received by the CAN controller, because the receive queue in the DPM and the CAN controller's receive buffer were overrun.
- CAN controller entered bus-off state.
- The firmware has detected an uncaught exception.

With the first two causes of an interrupt the respective event is set to the "Signalled state". As explained above, the functions HiCOCANRead, HiCOCANReadEX and HiCOCANWrite continue to process, reading the message from or writing it to the DPM.
With the other causes of an interrupt a driver-internal signalling flag is set. This flag causes the functions with the "can" parameter to return HTX_BUSOFF, HTX_OVERRUN or HTX_CANERROR in place of HTX_SUCCESS.

3.4 Interrupt Handling

3.4.1 Interrupt Handling in the Windows Driver

In the Windows versions of the driver the interrupt service routine performs all signalling tasks. **The application does not have to carry out an action. The following will only explain the driver-internal process.**

3.4.1.1 Receive Interupt

With the receive interrupt the event is set to "Signalled state". In this way, the HiCOCANRead or HiCOCANReadEX function is signalled that a received message is in the DPM.

3.4.1.2 Transmit Interrupt

With the transmit interrupt the interrupt service routine also sets a driver-internal event to "Signalled state". In this way, the HiCOCANWrite function is signalled that there is still space available in the corresponding transmit queue of the DPM.

3.4.1.3 Error Interrupts

As soon as an error interrupt occurs, the interrupt service routine sets a signalling event, which causes functions with the can parameter to return a value other than HTX_SUCCESS. Other return values can be: HTX_OVERRUN, HTX_BUSOFF or HTX_ERROR_SYSTEM.

3.5 Multitasking

The supplied driver is capable of multitasking, i.e., it is able to support more than one process at the same time. As a rule, all driver functions are available to all processes. However, there are some restrictions:

1. Each of the maximally 8 available CAN nodes may be used by one process only.

2. The HiCOCANReset function should not be used. This function resets all temporary parameter settings and deletes both the receive- and transmit queue. This will cause problems when the two CAN nodes of a board are used by several processes.

3. The HiCOCANSetTimestamp function should not be used, either. The timestamps of both nodes of a board are set by the same timer.
   If the two CAN nodes are used by several processes, and if the timestamps are analyzed each, the following problem occurs:
   When the HiCOCANSetTimestamp function is used, process 1 will increment the timestamp by one day. However, process 2 continues to receive messages. As a result, this process detects a one-day jump when evaluating the timestamp although only a few micro- or milliseconds have elapsed between the receipt of the two messages.

4. The HiCOCANResetDriver function usually re-enables all resources used by the driver. This is independent of the calling process, which means that a CAN node might not be accessed any more without restarting the DLL

3.6 General Functions

It is possible to have up to four boards in a system and they must differ in their board numbers (the relevant jumpers are described in later chapters) and their resources. But note; only ONE HiCOCAN-MiniPCI is supported due to the missing jumpers for the board number selection.

3.6.1 HiCOCANOpenDriver()

Before calling a driver's function for the first time, the driver has to be initialized. For this, call:

```
DWORD HiCOCANOpenDriver( void )
```

Return values:

HTX_SUCCESS
HTX_ERROR

Note
If this function yields HTX_ERROR, no further functions of the driver DLL may be invoked. Check whether the driver was installed properly.

3.6.2 HiCOCANCloseDriver()

Before closing the application call:

```
DWORD HiCOCANCloseDriver( void )
```

in order to re-enable all resources used by the DLL.

Return value:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR_SYSTEM

Note
After using the HiCOCANCloseDriver function, the HiCOCANOpenDriver function needs to be executed first, before you are able to use the driver's other functions!

Hint
Before calling this function, it is recommended that you halt all CAN nodes used by the calling process (application) via the HiCOCANStop function.
If this function was not used while debugging an application, for example, use the HiCOCANResetDriver function. This is necessary in order to free the resources used again.

3.6.3 HiCOCANResetDriver()

The function

```
DWORD HiCOCANResetDriver( void )
```

resets the driver, enabling all resources such as memory locations and interrupts.

Return values:

HTX_SUCCESS
HTX_ERROR_DRIVER
HTX_ERROR_APPLICATION

Important note

The HiCOCANResetDriver function should only be used during the development if the application was closed without successfully executing the HiCOCANCloseDriver function.

This function has an effect on all processes (applications) that make use of this driver! Therefore, make sure you have a look at section "Multitasking“ when using this function!

After using the HiCOCANResetDriver function, the HiCOCANOpenDriver function must be executed again!

3.6.4 HiCOCANGetErrorString()

Determines the symbolic name HTX_..... of the specified value:

```
DWORD HiCOCANGetErrorString (DWORD ErrorCode, TCHAR *Puffer)
```

Parameter	Description
`DWORD ErrorCode`	Error code for which the error text is to be determined.
`TCHAR *Puffer`	Pointer to the beginning of a buffer with a size of 100 characters, where the error code is to be entered.

Return values:

HTX_SUCCESS
HTX_ERROR

If the return value is HTX_ERROR, "UNKNOWN ERRORCODE" is returned as text in the array buffer.

This function is available for ANSI- and for Unicode. You may switch between the codes by defining the UNICODE symbol before linking the HICOCAN.h header file.

3.6.5 HiCOCANGetDriverInformation()

This function is used to query the version information. It has the following prototype:

```
DWORD HiCOCANGetDriverInformation( LPBYTE lpbVersionNumber,
                                   void* lpExtendedInformation );
```

Parameter	Description
`LPBYTE lpbVersionNumber`	Pointer to a variable in which the version number of the driver is to be entered.
`void *lpExtendedInformation`	Pointer to a structure of type sHiCOCANDriverInfo.

Return values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR

The function fills the specified variables with the corresponding information. The returned version number is coded as follows: The upper four bits contain the major version, and the lower four bits the minor version. This means that for driver version 4.5 the value 45h is returned.

With the lpExtendedInformation parameter the following has to be considered. The bStructVersion element needs to be initialized by the application with a valid value (for valid values, see the description of the sHiCOCANDriverInfo structure). If, after calling the function, this element is bStructVersion 0, then all other data in this structure are invalid.

3.7 Functions for Controlling the CAN Nodes

3.7.1 HiCOCANOpen()

This function is used to initialize the driver for a specific CAN node:

```
DWORD HiCOCANOpen( BYTE can  );
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.

Return values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR_DRIVER
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_MULTIPLE_NODE
HTX_ERROR_CANNOT_SET_IRQ
HTX_ERROR_REGISTRY
HTX_ERROR_ALREADY_OPENED
HTX_ERROR_ALREADY_USED
HTX_ERROR_BOARD_NOT_RUNNING
HTX_ERROR_CHECKSUM_BOOTSTRAPPER
HTX_ERROR_CHECKSUM_FIRMWARE
HTX_ERROR
HTX_ERROR_NOCONFIG
HTX_ERROR_SYSTEM

Executing the HiCOCANopen function informs the driver about the specified CAN node and performs the necessary initializations. After that, the CAN node may be accessed by all other functions.

3.7.2 HiCOCANClose()

The following function enables a CAN node which can then be used by another application:

```
DWORD HiCOCANClose ( BYTE can )
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.

Return values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

Note

- After using the HiCOCANCloseDriver function, the HiCOCANOpenDriver function needs to be executed first, before you are able to use the driver's other functions!
- In the Windows version HiCOCANClose is also called by the HiCOCANCloseDriver() function.

3.7.3 HiCOCANStart()

If HiCOCAN is at STOP state it can instantly re-enter RUN state with this option only:

```
DWORD HiCOCANStart ( BYTE can )
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

This function instructs HiCOCAN to activate the specified CAN controller, in order to establish the CAN message transfer. Available CAN messages in the DPM's message queues will be kept. In operating modes other than STOP the HiCOCANStart function has no effect.

3.7.4 HiCOCANStop()

The opposite function of HiCOCANStart is this function:

```
DWORD HiCOCANStop (BYTE can)
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

HiCOCANStop instructs the HiCOCAN firmware to set the specified CAN node to the reset mode, in order to stop the message transfer. Available CAN messages in the message queue of the DPM are deleted. In operating modes other than RUN HiCOCANStop has no effect.

3.7.5 HiCOCANReset()

The following function serves to restart the firmware on a HiCOCAN board:

```
DWORD HiCOCANReset ( BYTE boardnr )
```

Parameter	Description
BYTE boardnr	Entry specifying the HiCOCAN board by using one of the following definitions of the supplied HiCOCAN.H header file: #define HiCOCAN0 0 #define HiCOCAN1 2 #define HiCOCAN2 4 #define HiCOCAN3 6

Return values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_BOARD
HTX_ERROR_BOARD_NOT_RUNNING
HTX_ERROR_CHECKSUM_BOOTSTRAPPER
HTX_ERROR_CHECKSUM_FIRMWARE
HTX_ERROR_SYSTEM

Hint
Restarting the firmware will initialize HiCOCAN anew. As a consequence, possibly existing CAN messages will get lost.

When using this function, have a look at the "Multitasking" section!

3.7.6 HiCOCANResetContr()

The following function can be used to reset specific CAN nodes:

```
DWORD HiCOCANResetContr( BYTE can, bool newInit )
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.
bool newInit	If TRUE, the specified CAN node is initialized again with the configuration data from the flash.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

If the CAN node is not initialized again, it is restarted in any case. If the CAN node is initialized again, it will be started only if the corresponding configuration parameter StartMode is unequal to null. This information is obtained from the relevant configuration data contained in the flash.

Note
Not supported by HiCOCAN-MiniPCI!

3.7.7 HiCOCANClrOverrun()

The Overrun bit indicates that at least one message could not be received (and got lost) due to a full receive queue. The Overrun bit of a specific CAN node is reset with this function:

```
DWORD HiCOCANClrOverrun( BYTE can )
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

3.7.8 HiCOCANAbortTransmit()

Using the following function, a message that is currently being transferred by the CAN controller can be aborted:

```
DWORD HiCOCANAbortTransmit( BYTE can )
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

Note
Not supported by HiCOCAN-MiniPCI!

3.7.9 HiCOCANRegisterEvent()

Using the HiCOCANRegisterEvent function, a Win32 event can be registered, which is set when the specified level of the receive queue is reached or exceeded.

The function has this prototype:

```
DWORD HiCOCANRegisterEvent(   BYTE can, HANDLE *lphEvent,
                              DWORD dwTriggerLevelSet,
                              DWORD dwTriggerLevelReset );
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.
HANDLE *lphEvent	Pointer to the handle of the Win32 event
dwTriggerLevelSet	Trigger level at which the Win32 event is set
dwTriggerLevelReset	Trigger level at which the Win32 event is reset

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM
HTX_ERROR_TRIGGERLEVELSET_INVALID
HTX_ERROR_TRIGGERLEVELRESET_INVALID

The application creates a Win32 event and passes the pointer to the handle of this function. The driver sets the specified event by means of the OS function SetEvent if the level of the receive queue reaches and/or exceeds the specified trigger level dwTriggerLevelSet (number of messages in the receive queue >= dwTriggerLevelSet). The driver resets the specified event when the specified trigger level dwTriggerLevelReset is not reached (number of messages in the receive queue < dwTriggerLevelReset).

Passing the value NULL in the lphEvent parameter to the HiCOCANRegisterEvent function will again deactivate this functionality.

Note

The driver expects the event to be passed to be valid. This means that the calling application cancels the event for the driver and then returns the handle to the operating system via CloseEvent.

Example using the function HiCOCANRegisterEvent :

This example demonstrates the use of the function HiCOCANRegisterEvent for can node 0. In this example, the application wants the event to be set when 20 or more messages in the receive queue of the can node 0.

The application also tells the driver that the event is to be reset when the application has read the messages and there are less than five messages in the receive queue of can node 0.

```
HANDLE      hEvent;
DWORD       dwRetVal = HTX_SUCESS;

// Create Win32 event. IMPORTANT : You MUST create a manual reset event,
// which is not signaled !!!
hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
if (hEvent == NULL)
{
      // ERROR : You MUST NOT call HiCOCANRegisterEvent, because the
      // event is not valid !
      // error handling
      return;
}
dwRetVal = HiCOCANRegisterEvent( 0, &hEvent, 20, 5 );
if ((dwRetVal != HTX_SUCCESS) &&
   (dwRetVal != HTX_OVERRUN) &&
   (dwRetVal != HTX_BUSOFF) &&
   (dwRetVal != HTX_CANERROR) )
{
      // ERROR : The event will NOT handled by the HiCOCAN driver. Check
      // the error code for detail information.
      // error handling
      return;
}
// IMPORTANT : You MUST NOT close the event handle before you have the
// event deregistered by the driver (see below)

// begin a loop
while ( fEndApplication == FALSE )
{
      // Wait until the event is signaled
      dwRetVal = WaitForSingleObject( hEvent, INFINITE );
      if ( dwRetVal == WAIT_OBJECT_0 )
      {
            // event signal -> do something e.g. read telegrams by calling
            // HiCOCANRead or HiCOCANReadEx
      }
      else
      {
            // error handling
      }
}
```

```
// We want that the application should exit -> deregister event
dwRetVal = HiCOCANRegisterEvent( 0, NULL, 0, 0 );
if ((dwRetVal != HTX_SUCCESS) &&
   (dwRetVal != HTX_OVERRUN) &&
   (dwRetVal != HTX_BUSOFF) &&
   (dwRetVal != HTX_CANERROR) )
{
      // ERROR : The event will NOT handled by the HiCOCAN driver.
      // Check the error code for detail information.
      // error handling
      return;
}
// This is the earliest-possible moment to close the event handle !!!.
CloseHandle( hEvent );
```

3.7.10 HiCOCANCanCtrl()

This function is only supported by the PCI based boards (HiCOCAN-PCI and HiCOCAN-MiniPCI). This function allows setting the board into a specific operation mode depending on the command value.

```
DWORD HiCOCANCanCtrl (BYTE can, BYTE cmd)
```

Parameter	Description
BYTE can	Number of the CAN node for which the driver is to be opened.
HANDLE *lphEvent	Pointer to the handle of the Win32 event
dwTriggerLevelSet	Trigger level at which the Win32 event is set
dwTriggerLevelReset	Trigger level at which the Win32 event is reset

Supported command	Description
HiCOCAN_SET_PASSIV	Setting the can node to passive mode.
HiCOCAN_EN_AUTO_OVERR	Clearing overrun is done by the firmware
HiCOCAN_DIS_AUTO_OVERR	Clearing overrun must be done by the application

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

3.8 Timestamp

HiCOCAN's firmware provides each message with a timestamp indicating the time (with 10-µs resolution) when the message was received. The timer installed on the board starts with power-on or every time the board is reset at 0 days, 00:00:00.000.000 hours.

3.8.1 HiCOCANSetTimestamp()

This function enables the application to set a timestamp at a specific time: The HiCOCANSetTimestamp function has been implemented to enable the application to place timestamps at a particular time. It provides the firmware with the specified timestamp information. The firmware will then accordingly set the timer on the HiCOCAN module.

```
DWORD HiCOCANSetTimestamp( BYTE board, psTS timestamp )
```

Parameter	Description
BYTE board	Entry specifying the HiCOCAN board by using one of the following definitions of the supplied HiCOCAN.H header file: #define HiCOCAN0 0 #define HiCOCAN1 2 #define HiCOCAN2 4 #define HiCOCAN3 6
psTS timestamp	Pointer to a variable of the type sTS containing the relevant time information. The sTS data type is defined in the supplied C-header file HiCOCAN.h; see chapter “Data Structures”.

Return values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_BOARD
HTX_ERROR_SYSTEM

Notes:

1.) With HiCOCAN-MiniPCI it is only possible to reset the timestamp counter to zero!

2.) Make sure you have a look at section "Multitasking“ when using this function!

3.9 Read/Write Functions

3.9.1 HiCOCANWrite()

HiCOCANWrite instructs the driver layer to write a CAN message to the transmit queue of the corresponding CAN node. The data for the structure of the CAN message must be reported in form of a pointer to a structure of the type sPCCanMsg. This type is defined in the supplied HiCOCAN.h C-header file; see chapter "Data Structures".

```
DWORD HiCOCANWrite( BYTE Can, psPCCanMsg msg, DWORD Timeout )
```

Parameter	Description
BYTE can	Number of the CAN node which is used to transmit the message.
psPCCanMsg msg	Pointer to the message data
DWORD Timeout	Specifies the time HiCOCANWrite is waiting for a free entry in the transmit queue when the transmit queue is full. The parameter is entered in 100ns units. The value 0 specifies an infinite waiting period.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_FULL_QUEUE
HTX_ERROR_ILLEGAL_ID
HTX_ERROR_ILLEGAL_LENGTH
HTX_ERROR_SYSTEM

HiCOCANWrite first checks whether there is sufficient space in the transmit queue of the desired CAN node. If sufficient space is available, the CAN message is created from the specified message data and written to the transmit queue. If the space is unavailable, the HTX_ERROR_FULL_QUEUE error code is immediately returned with the DOS version. With the Windows version the application is waiting until the timeout time has elapsed. If there is no free entry in the transmit queue until the timeout time has elpased, the Windows version will also return the HTX_ERROR_FULL_QUEUE error code.

Hint
The timestamp data are taken into consideration.

3.9.2 HiCOCANRead()

Using the driver's HiCOCANRead function, a CAN message can be read from the receive queue of a specific CAN node. HiCOCANRead enters the message data in a structure of the type sPCCanMsg. A corresponding structure needs to be provided by the application (allocated memory), and HiCOCANRead must have a pointer to this structure. The sPCCanMsg type is defined in the supplied HiCOCAN.h C-header file; see section "Data Structures".

```
DWORD HiCOCANRead( BYTE can, psPCCanMsg msg, DWORD Timeout )
```

Parameter	Description
BYTE can	Number of the CAN node which received the message.
psPCCanMsg msg	Pointer to the message data
DWORD Timeout	Specifies the time HiCOCANRead is waiting for the entry of new messages when the receive queue is empty. The parameter is entered in 100ns units. The value 0 specifies an infinite waiting period.

Return values:

- HTX_SUCCESS
- HTX_OVERRUN
- HTX_BUSOFF
- HTX_CANERROR
- HTX_ERROR_APPLICATION
- HTX_ERROR_UNKNOWN_NODE
- HTX_ERROR_EMPTY_QUEUE
- HTX_ERROR_SYSTEM

HiCOCANRead first checks whether there is a message available in the receive queue of the desired CAN node. If this is the case, the message is transferred to the structure provided by the application to which the msg pointer points. The supplied timestamp specifies the time that the CAN controller received this message. For more detailed information, please refer to section "HiCOCANSetTimestamp()".

If no message is available in the corresponding receive queue in the DOS driver, HTX_ERROR_EMPTY_QUEUE is returned.

The Windows version of the driver waits until the timeout time has elapsed. If no message is received during the timeout time, the error code HTX_ERROR_EMPTY_QUEUE is returned; otherwise the received message is returned.

3.9.3 HiCOCANReadEx()

Using the driver's HiCOCANReadEx function, a list of CAN message can be read from the receive queue of a specific CAN node. HiCOCANReadEx enters the list of message data in a structure of the type sPCCanMsg. A corresponding structure needs to be provided by the application (allocated memory), and HiCOCANReadEx must have a pointer to this structure. The sPCCanMsg type is defined in the supplied HiCOCAN.h C-header file; see chapter "Data Structures" on p. **Fehler! Textmarke nicht definiert.**. ListSize is a pointer to the size of the list. When the function returns, it contains the actual number of CAN telegrams read.

```
DWORD HiCOCANReadEx(    BYTE can, psPCCanMsg MsgList, DWORD Timeout,
                        DWORD *ListSize )
```

Parameter	Description
BYTE can	Number of the desired CAN node.
psPCCanMsg MsgList	Pointer to the message data
DWORD Timeout	Specifies the time HiCOCANReadEx is waiting for the entry of new messages when the receive queue is empty. The parameter is entered in 100ns units. The value 0 specifies an infinite waiting period.
DWORD ListSize	Size of the list, provided by the application. When HiCOCANReadEx return this variable contains the number of entries returned.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_EMPTY_QUEUE
HTX_ERROR_SYSTEM

HiCOCANReadEx first checks whether there are messages available in the receive queue of the desired CAN node. If this is the case, the messages are transferred to the list by the application to which the MsgList pointer points. The supplied timestamp specifies the time that the CAN

controller received this message. For more detailed information, please refer to section "HiCOCANSetTimestamp()".

The Windows version of the driver waits until the timeout time has elapsed. If no message is received during the timeout time, the error code HTX_ERROR_EMPTY_QUEUE is returned; otherwise the received message is returned.

3.10 Status Request

3.10.1 HiCOCANState()

The state of a HiCOCAN board can be requested with this function:

```
DWORD HiCOCANState( BYTE board, BYTE* State )
```

Parameter	Description
BYTE board	Entry specifying the HiCOCAN board by using one of the following definitions of the supplied HiCOCAN.H header file: #define HiCOCAN0 0 #define HiCOCAN1 2 #define HiCOCAN2 4 #define HiCOCAN3 6
BYTE *State	Pointer to a variable of type BYTE defined by the calling function. The state of desired board is written to the specified variable. Table 2 displays the significance of the returned value. Please note that the HiCOCAN boards is at the “ready” state only, if all status bits are 0 except for status bit 6.

Return values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_BOARD
HTX_ERROR_SYSTEM

Hint
The returned state is valid only if the function returns HTX_SUCCESS.

Bit pos.	Function	Bit Value	Significance
7	ERROR	1 = yes	General error
6	RUN	1 = yes	Firmware running
5	SYS_FAIL	1 = yes	Hardware error
4	FW_CHK	1 = yes	Firmware error (checksum)
3	BOOTCHK	1 = yes	Bootstrapper error (checksum)
2	---		Reserved
1	CFGCHK	1 = yes	No configuration data available for the CAN nodes
0	---		Reserved

Table 2: Significance of the returned value

3.10.2 HiCOCANStateTrans()

The state of the fault tolerant error line of the HiCOCAN board can be requested with this function:

```
DWORD HiCOCANState( BYTE board, BYTE* State )
```

Parameter	Description
BYTE board	Entry specifying the HiCOCAN board by using one of the following definitions of the supplied HiCOCAN.H header file: #define HiCOCAN0 0 #define HiCOCAN1 2 #define HiCOCAN2 4 #define HiCOCAN3 6
BYTE *State	Pointer to a variable of type BYTE defined by the calling function. The state of the fault tolerant transceiver is written to the specified variable. If bit 0 is set to 0, everything is ok and if bit 0 is 1 a fault tolerant occurred.

Return values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_BOARD
HTX_ERROR_SYSTEM

Hint

The returned state is valid only if the function returns HTX_SUCCESS.

3.10.3 HiCOCANStateContr()

The HiCOCANState function provides information on the state of the CAN controller:

```
DWORD HiCOCANStateContr( BYTE can, BYTE* State, BYTE* NumOfRecErrors,
                         BYTE* NumOfTraErrors )
```

Parameter	Description
BYTE can	Number of the desired CAN node.
BYTE *State	Pointer to a variable of type BYTE defined by the calling function. The state of the desired CAN controller is written to the specified variable. Table 3 displays the significance of the returned value.
BYTE *NumOfRecErrors	Pointer to a variable of type BYTE defined by the calling function. The number of the receive errors is written to the specified variable.
BYTE *NumOfTraErrors	Pointer to a variable of type BYTE defined by the calling function. The number of the transmit errors is written to the specified variable.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

Hint

The returned state and the numbers of errors are valid only if the function returns HTX_SUCCESS.

Bit pos.	Function	Bit Value	Significance
7	Bus-Off	1 = yes	CAN controller is in the Bus off state
6	Error Passive	1 = yes	CAN controller is in the Error-Passive state
5	Controller transmits a message	1 = yes	
4	Controller receives a message	1 = yes	
3	Last request for transmit message successfully terminated	1 = yes	
2	Transmit buffer available	1 = yes	
1	Overrun CANRxFifo	1 = yes	CAN controller has set the Overrun bit, i.e., at least one message could not be received due to a full receive queue
0	Messages are available	1 = yes	A message is available in the receive buffer of the controller

Table 3: Significance of the returned value

Note
If the status bits 4 and 5 are simultaneously set, the controller is in the Stop state.
For a more detailed description of the significance, please refer to the CAN controller's data sheet [12] or the Application Note [13].

3.10.4 HiCOCANTraQState(), HiCOCANRecQState()

Using the functions

```
DWORD HiCOCANTraQState( BYTE can )
```

```
DWORD HiCOCANRecQState( BYTE can )
```

the state of the transmit or receive queue of each CAN node can be examined. If no error occurred, the number of entries is returned.

Parameter	Description
BYTE can	Number of the desired CAN node.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM
HTX_TRANSMITQUEUE_FULL (only for function HiCOCANTraQState)
HTX_RECEIVEQUEUE_EMPTY (only for the function HiCOCANRecQState)
Number of the messages entered

3.11 Modifying the Communication Parameters

The functions described in this chapter allow you to temporarily alter communications parameters such as the baud rate or acceptance filter.

3.11.1 HiCOCANSetAcceptMask()

The CAN controller allows for filtering the messages by means of the hardware. For this purpose, several registers are provided whose functions are described in the data sheet [12] or the corresponding Application Note [13], respectively.

The acceptance mask is set with this function:

```
DWORD HiCOCANSetAcceptMask(   BYTE can, BYTE FilterMode,
                              DWORD Code,DWORD MaskReg)
```

Parameter	Description
BYTE can	Number of the desired CAN node.
BYTE FilterMode	Value of the acceptance filter mode; As for the significance of this value, please refer to the data sheet [12] or the Application Note [13]. The constants HiCOCAN_FILTERMODE_DUAL and HiCOCAN_FILTERMODE_SINGLE defined in the HiCOCAN.h header file can be used to set this value.
DWORD Code	Value of the acceptance code register; The most significant byte with the register ACR3 and the least significant byte with register ACR0 correspond with each other. As for the significance of this value, please refer to the data sheet [12] or the Application Note [13].
DWORD MaskReg	Value of the acceptance register; The most significant byte with the register ACR3 and the least significant byte with register ACR0 correspond with each other. As for the significance of this value, please refer to the data sheet [12] or the Application Note [13].

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

Note
Not supported by HiCOCAN-MiniPCI!

3.11.2 HiCOCANSetTimingReg(), HiCOCANSetTimingRegEx(), HiCOCANSetBaud()

The baud rate at which the CAN node operates is determined by the two 8-bit-wide bus timing registers BTR0 and BTR1. As for the significance of the contained bits, please refer to the data sheet [12] or the Application Note [13] of the CAN controller used. Note that the CAN controllers are operated with 20 MHz.

Two functions are provided for setting these registers and thus the baud rate. The first function writes the specified values for the Bus timing registers of the specified CAN node directly to these registers:

```
DWORD HiCOCANSetTimingreg( BYTE can, BYTE reg0, BYTE reg1 )
```

Parameter	Description
BYTE can	Number of the desired CAN node.
BYTE reg0	Value for timing register 0
BYTE reg1	Value for timing register 1

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

This function is not supported by HiCOCAN-MiniPCI. Please use the function:

```
DWORD HiCOCANSetTimingregEx( BYTE can, DWORD btr )
```

Parameter	Description
BYTE can	Number of the desired CAN node.
DWORD btr	Value for timing register

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

The second function:

```
DWORD HiCOCANSetBaud( BYTE can, BYTE rate )
```

takes the values for the bus timing register from the Board's flash device. These values are stored by means of the configuration tool. The parameters to be specified are the following:

Parameter	Description
BYTE can	Number of the desired CAN node.
BYTE rate	Entry specifying the desired baud rate by using the following definitions (defined in the HiCOCAN.h header file): #define HiCOCAN_BAUD10K 1 #define HiCOCAN_BAUD20K 2 #define HiCOCAN_BAUD50K 5 #define HiCOCAN_BAUD100K 10 #define HiCOCAN_BAUD125K 12 #define HiCOCAN_BAUD250K 25 #define HiCOCAN_BAUD500K 50 #define HiCOCAN_BAUD1M 100

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR

HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_SYSTEM

Important note
A message that might be transmitted while the baud rate is being changed will get lost with both functions!

3.11.3 HiCOCANParameter()

The HiCOCANParameter function allows you to determine the communications parameters currently set in the CAN controller:

```
DWORD HiCOCANParameter( BYTE can, BYTE *TimingReg0, BYTE *TimingReg1,
                        WORD *Baudrate, BYTE *FilterMode, DWORD *Code,
                        DWORD *MaskReg)
```

Parameter	Description
BYTE can	Number of the desired CAN node.
BYTE* TimingReg0, BYTE* TimingReg1	Pointer to two variables to which the values of the bus timing registers 0 and 1 are copied.
WORD* Baudrate	Pointer to a variable which is provided with a value (in kBaud) for the baud rate.
BYTE* FilterMode, DWORD *Code, DWORD *MaskReg	Pointer to variables where the values for acceptance filter mode, the acceptance code register and the acceptance mask register are to be entered.

Return values:

HTX_SUCCESS
HTX_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_NOT_SUPPORTED
HTX_ERROR_SYSTEM

Hint

The parameters' returned values are valid only if the function returns HTX_SUCCESS.

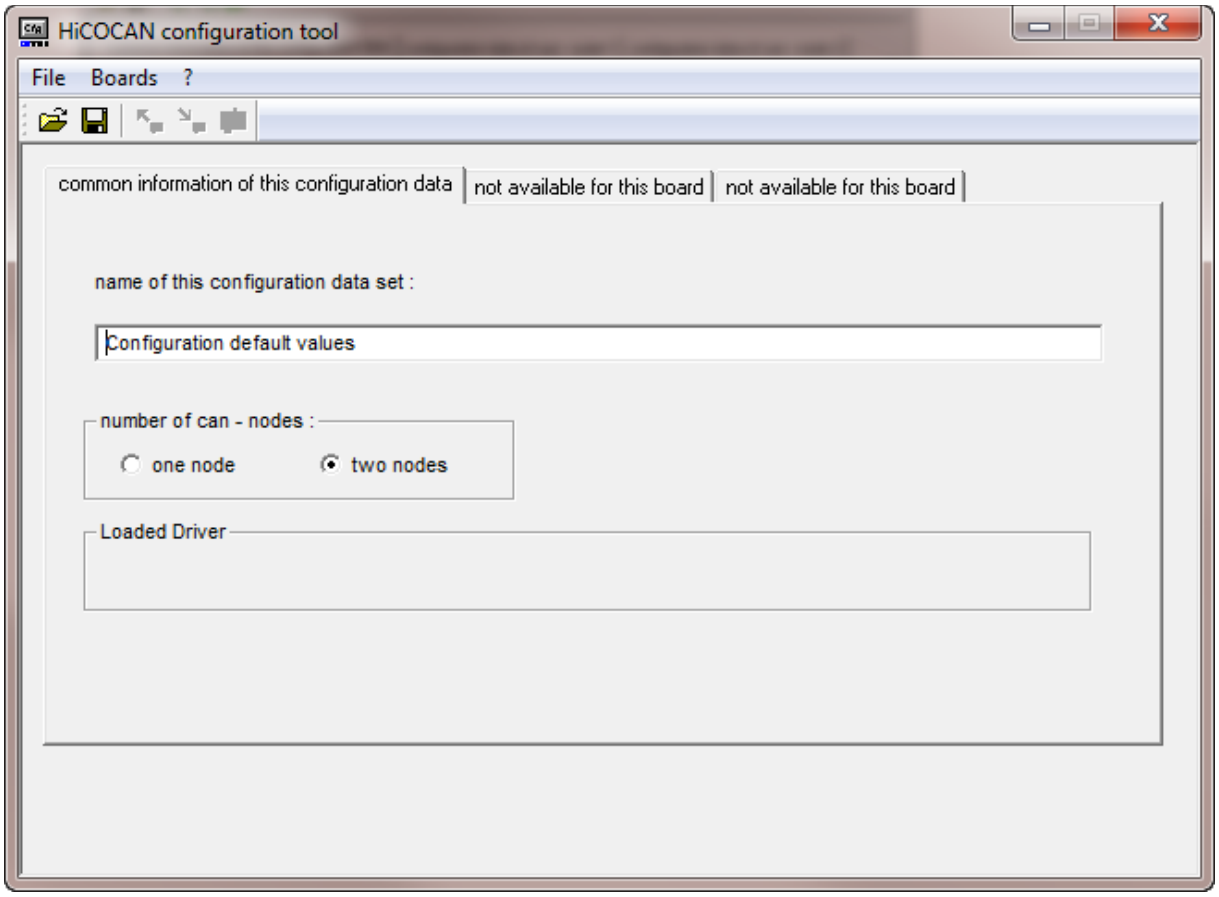
Note

Not supported by HiCOCAN-MiniPCI!

4 Configuration Tool under Windows

The configuration tool serves to permanently configure the board. It is assumed that the driver for the boards has been installed prior to starting up the system.

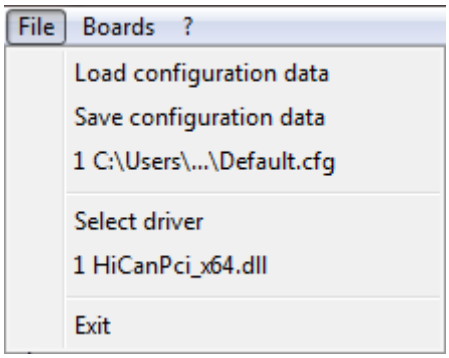
When starting the configuration tool, the following dialog box appears:



Important:
On Windows x64 the 64-bit edition of the configuration tool is installed. The 64-bit edition **must** use the 64-bit version of the driver DLL. See section "Configuration Tool and CAN Monitor on Windows x64" for more information.

4.1 The Menus and Buttons

4.1.1 File Menu



The "Select driver" menu item selects the driver for the board. The DLL name depends on the board type and if the OS is a 32-bit or 64-bit OS:

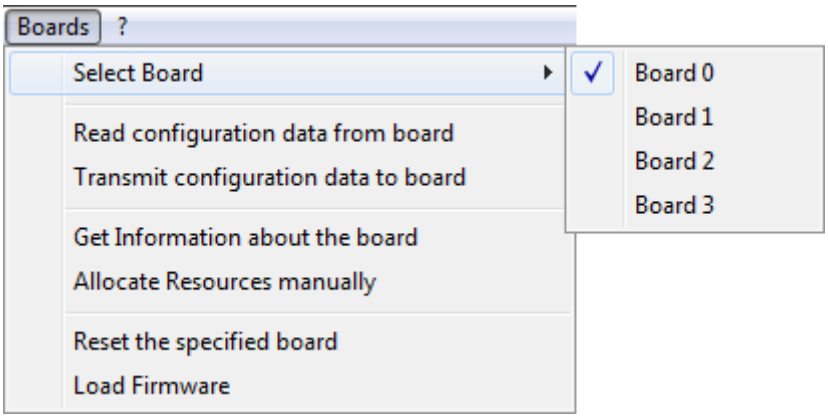
Board type	32-bit OS	64-bit OS
HiCOCAN-ISA / HiCOCAN-104	HiCOCAN3.DLL	---
HiCOCAN-PCI / HiCOCAN-CPCI	HiCanPci.DLL	HiCanPci_x64.DLL
HiCOCAN-MiniPci (-2CH/-4CH)	HiCanMiniPci.DLL	HiCanMiniPci_x64.DLL
HiCOCAN-PCI104	HiCanPci104.DLL	HiCanPci104_x64.DLL

The driver DLLs can be found in the System or System32 folder of your operating system.

The "load configuration data" menu item loads configuration data from a file into the input mask. As for the format of the file to be read, please refer to the section “Format of the configuration Files”.

The "save configuration data" menu item saves the configuration data currently displayed in the input mask to a configuration file.

4.1.2 Boards Menu



Via the "Select Board" menu item you can select the board to be accessed with the other functions of this menu. The currently selected board is indicated by a check mark to the left of its name.

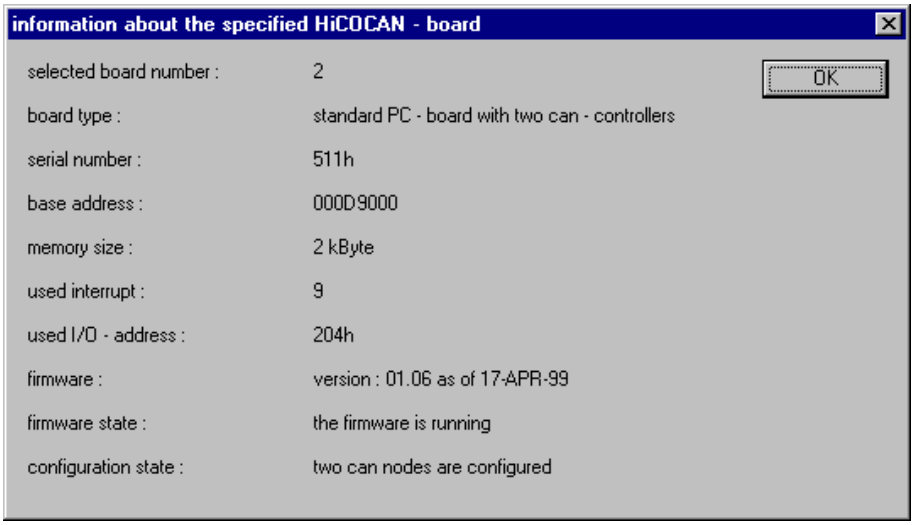
Remark
If the HiCOCAN-MiniPCI driver DLL is used all items in the “Select Board” menu are disabled and the board number is fixed to zero. This is done due to fact that only one HiCOCAN-MiniPCI board can be in the system.

The menu item "Read configuration data from board" reads the configuration data currently saved in the flash memory of the selected board and displays them in the input mask.

Hint
When selecting the "Read configuration data from board" menu item, the configuration data previously entered in the input mask will be deleted!

The menu item "Transmit configuration data to board" transmits the data displayed in the input mask to the selected board. The firmware saves the configuration data in the board's flash memory. Any previously saved data will get lost.

The "Get information about the board" menu item displays information on the currently selected board. (Here, information on the HiCOCAN-104 board is displayed.)



Label	Meaning
Selected board number	Adjusted/assigned board number
Board type *)	The following types are possible: • PC104 card with two CAN controllers • PC104 card with one CAN controller • Standard PC card with two CAN controllers • Standard PC card with one CAN controller • HiCOCAN-104 second generation
Serial number *)	Serial number of the card entered in the Plug and Play controller
Base address *)	The assigned base address in main memory
Memory size *)	Size of the address space used by the board
Used interrupt *)	The interrupt allocated to the board
Used I/O-address *)	The I/O address allocated to the board
Firmware	Version number and date of revision of the firmware
Firmware state	Status messages of the firmware in plain text
Configuration state	Number of currently configured can node's on the board

*) This information is displayed for PC/104- and ISA boards only.

Using the "Allocate Resources manually" menu item, you are able to selectively allocate resources to the HiCOCAN boards (address of the DPM, I/O address and interrupt). This can also be done when no resources were allocated by the Plug and Play mechanism of the BIOS or operating system. This menu item can be used in order to enter the resources for HiCOCAN-104-2H, HiCOCAN-104-2L of the HiCOCAN board into the registry.
This menu item is of no significance with the HiCOCAN-PCI/HiCOCAN-CPCI boards.

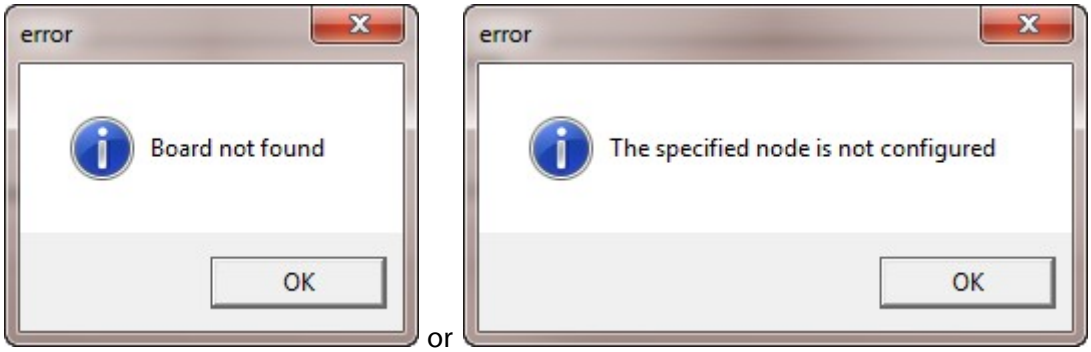
The "Reset the specified board" menu item resets the desired board; that is, the board's firmware is started anew, as is the case when switching on the computer.

Using the "Load Firmware" menu item, the firmware of the board may be exchanged.

For HiCOCAN-104/PCI/CPCI S-Records are used and for HiCOCAN-MiniPCI a binary format of the firmware is used.

Note

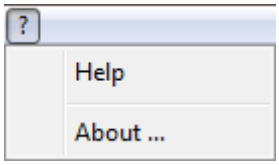
If the selected board is not available in the system when selecting the menu items "Read configuration data from board", "Transmit configuration data to board", "Get information about the board" or "Reset the specified board", one of the following error message displays:



If the error message appears although the board was installed and has a correctly assigned board number, this may be due to the following:

- No resources (address for the DPM, I/O address and interrupt) were allocated to the board through the Plug and Play mechanism.
- The resources (address for the DPM, I/O address and interrupt) for a non-Plug and Play card have not been written to the registry

4.1.3 Help Menu



The "Help" menu item will call the configuration tool's online help system. Since the online help is an HTML file, you will need a browser to view the file.

The "About..." menu item provides information on the configuration tool's version number.

4.1.4 The Symbol Bar

The symbol bar provides various menu commands for fast access:

- Load configuration data: 
- Save configuration data: 
- Read configuration data from board: 
- Transmit configuration data to board: 
- Get information about the board: 

4.2 Format of the Configuration Files

The structure of a configuration file corresponds to that of a Windows .INI file. This allows you to create a configuration file with the configuration tool or by means of any editor.

The configuration file consists of the following three sections: HiCOCAN, CAN-NODE1 and CAN-NODE2, which are described below.

4.2.1 The HiCOCAN Section

The HiCOCAN section contains the parameters which are valid for the entire board. The **recordname** entry is the name of the data set. The second entry **numChann** specifies the number of the configured CAN nodes.

4.2.2 The Sections CAN-NODE1 and CAN-NODE2

The sections CAN-NODE1 and CAN-NODE2 contain the parameters which apply to the nodes 1 and 2. These sections are not supported for HiCOCAN-MiniPCI. It is up to the application to program the correct settings for the system. These parameters are determined by the following entries:

Entry	Meaning
StartMode	If the parameter is 0, the CAN node has to be started by the application using the **HiCOCANStart** function. If it is unequal 0, the CAN node starts automatically after power on
btr0	Specifies the value of the bus timing register 0, which is written to the SJA1000 controller during initialization. The value to be specified is a two-digit hex number. For mor details, please refer to the data sheet [12] or the Application Note [13] of the CAN controller.
btr1	Specifies the value of the bus timing register 1, which is written to the SJA1000 controller during initialization. The value to be specified is a two-digit hex number. For more details, please refer to the data sheet [12] or the Application Note [13] of the CAN controller.
afm	Specifies the value of the acceptance filter mode register, which is written to the SJA1000 controller during initialization. The value to be specified is a two-digit hex number. For more details, please refer to the data sheet [12] or the Application Note [13] of the CAN controller

Entry	Meaning
acr	Specifies the value of the acceptance code register, which is written to the SJA1000 controller during initialization. The value to be specified is an eight-digit hex number. For more details, please refer to the data sheet [12] or the Application Note [13] of the CAN controller.
amr	Specifies the value of the acceptance mask register, which is written to the SJA1000 controller during initialization. The value to be specified is an eight-digit hex number. For more details, please refer to the data sheet [12] or the Application Note [13] of the CAN controller.
btr_10k, btr_20k, btr_50k, btr_100k, btr_125k, btr_250k, btr_500k, btr_1M	These parameters serve to specify the values for the baud rate for the bus timing registers 0 and 1. The values are written to the CAN controller's registers with the **HiCOCANSetBaud** function. The value to be specified is a four-digit hex number, where the first two digits represent the value for bus timing register 0 and the last two digits the value for bus timing register 1.

5 Sample Application

In order to show you how to use the application interface, the software package includes two sample applications. The sample application hicocandemo.exe is for boards with two CAN nodes, the sample application hicocandemo_4ch.exe is for HiCOCAN-MiniPci-4ch. These applications transmit CAN messages between the two resp. four CAN nodes of a board, which are connected via a cable for this purpose.

These applications are build in a Visual Studio 2008 environment.

Remark for Windows CE:
The application cannot be compiled within the platform builder environment. It's a separate VS2008 based C++ application which includes statically the MFC libraries. Before compiling, it is necessary to install the SDK for the used kernel.

5.1 Detailed Description

After starting the application, the required settings are taken for using the two CAN nodes:

1. Open the driver
2. Open the nodes
3. Start the nodes

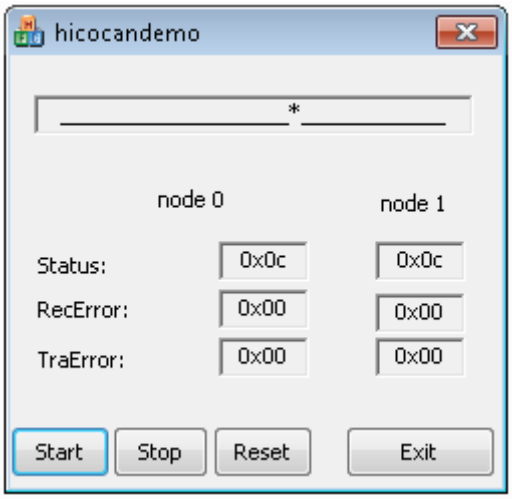
After this, three working threads will be created. Two of these threads each serve one CAN node. The message will then be sent between these CAN nodes in a "ping-pong" manner.

For this, the first thread ("PingThread") initiates the transfer by creating a message and sending it via the first node.

The second thread ("PongThread") receives this message via the second CAN node, modifies the identifier and retransmits the message via the second CAN node.

The PingThread receives the message from the first node, and updates the output items of the dialog.

The PingThread retransmits the message via the first CAN node to the PongThread, which is already waiting for an incoming message.



The third Thread cyclically polls the status of the two nodes and displays the information obtained in the application window.

The program will be executed until the user closes the window. A set flag causes the threads to close themselves. In addition, the driver is closed and the program left.

Remark
The sample application hicocandemo_4ch is the same for a board HiCOCAN-MiniPCI-4CH and contains two additional threads for the additional CAN nodes.

6 Troubleshooting (HiCOCAN-xxx)

6.1 Support

This product has been thoroughly tested over the development period. Due to its complexity, however, no guarantee can be given that the boards operate seamlessly under any circumstances. We are therefore grateful for any feedback regarding an improper operation of the boards.

If any problems occur, have a look at the FAQ section of this manual first. Or visit our website at http://www.support.emtrion.com for the latest FAQ.

If you cannot find the necessary information, contact our Support Team via e-mail, fax or phone. Your support question will be answered as soon as possible.

To accelerate the process, please fill out the supplied form, which can be found in the Support directory of the CD or on the internet at http://www.emtrion.com/support_form_en.php.

Please fill in the form and send, fax or email it to:

Emtrion GmbH
Alter Schlachthof 45
D-76131 Karlsruhe
Tel: 0721 / 62725 – 0
Fax: 0721 / 62725 – 19
Email : support@emtrion.de

7 Reference

[1] PC/104 Specification
Version 2.3, June 1996
http://www.controlled.com/pc104/techp1.html

[3] Am29F400B, CMOS 5.0V only, sector erase Flash Memory
AMD

[4] CY7C132, 2Kx8 Dual-Port Static RAM
Cypress Semiconductor Corporation

[10] Plug-and-Play BIOS Specification
Version 10.A, May 5, 1994
Compac Computer Corporation, Phoenix Technologies Ltd., Intel Corporation

[11] Extended System Configuration Data Specification
Version 1.0A, May 31, 1994
Compac Computer Corporation, Phoenix Technologies Ltd., Intel Corporation

[12] SJA1000 Stand-alone CAN controller
Data Sheet
Preliminary specification, 1997 Nov 04
Philips Semiconductors

[13] Application Note SJA1000 Stand-alone CAN controller
AN97076
Authors : Hank, Peter and Jöhnk Egon
1997 Dec 15
Philips Semiconductors

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP